Vol. 4, No. 6 November/December 1986 $4.00

programmer's journal

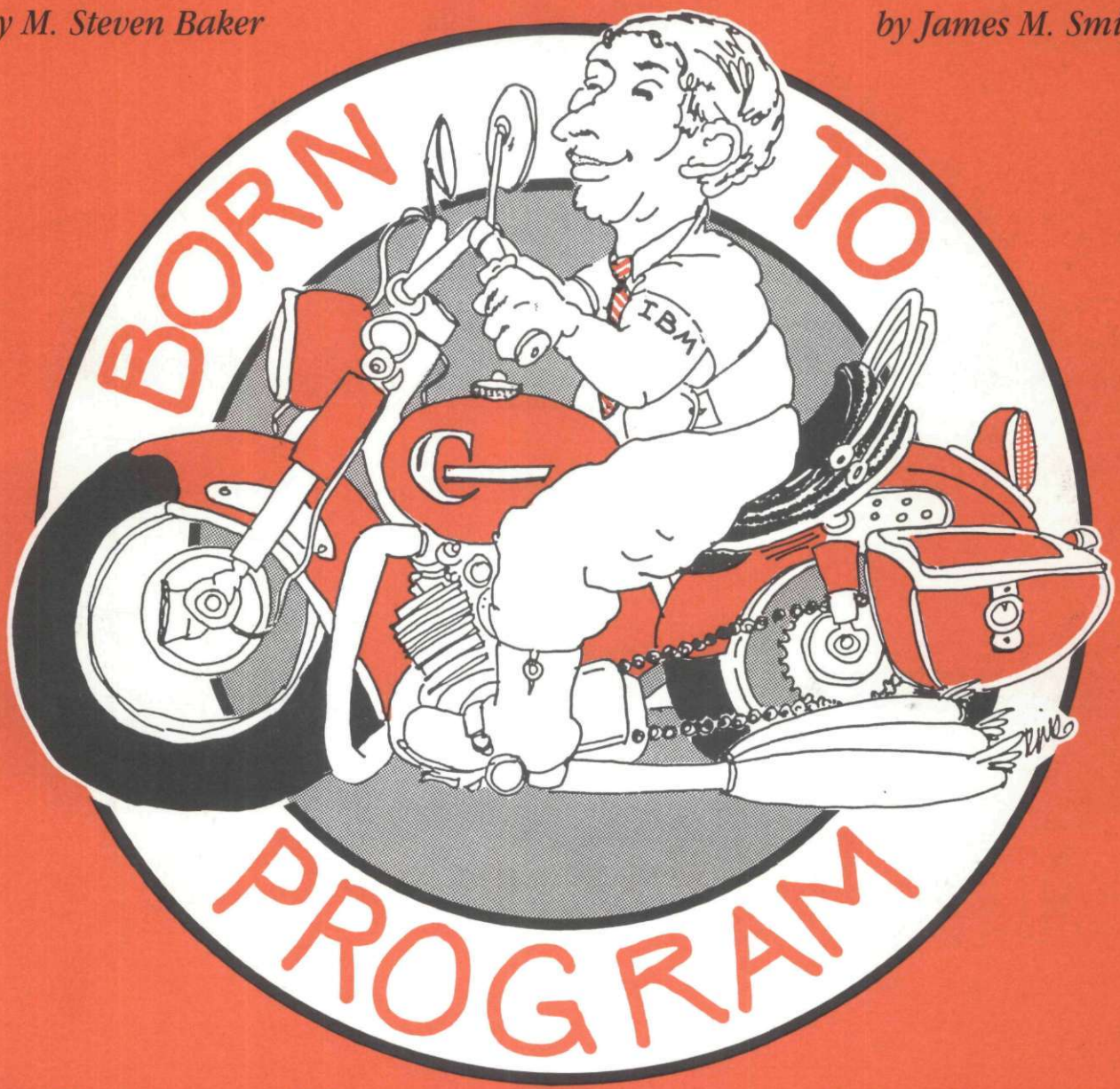
The Resource Journal for IBM PC Programmers

Terminate and Stay Resident

by M. Steven Baker

Turbo Pascal Inline Code

by James M. Smith



Genifer: A Powerful dBASE Applications Generator

by Paul Barkley

Advanced Trace86: a Debugger and More

by Peter Feldman

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

ACTOR WILL DO WINDOWS.

If you've been wondering how you're going to explore Microsoft® Windows and then work it into your programs, what you need has just arrived.

ACTOR™

ACTOR is a new programming system, and there are two reasons why it's so good with Windows.

First, ACTOR is an ideal programming environment. It shows you all your work through Windows. Any part of the program you're writing, its output as it runs, error diagnostics and tracing, catalogs of routines, and, in fact, just about anything you like—they can all be visible on the screen and accessible at the same time, in their own windows.

So it's easier to see what you're doing.

Second, ACTOR is an object-oriented programming language. Everything in an ACTOR program is a self-contained package with its own data and procedures. That is, an object. And the objects pass messages to each other while the program runs.

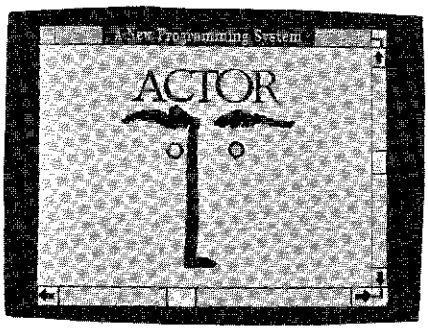
With ACTOR, you don't write lists of instructions. Instead, you define what classes of objects do. Which is especially easy because they inherit most of their behavior from other classes. That means there's less for you to worry about.

And when you create an object, you just choose its name and starting values. After that it pretty much takes care of itself.

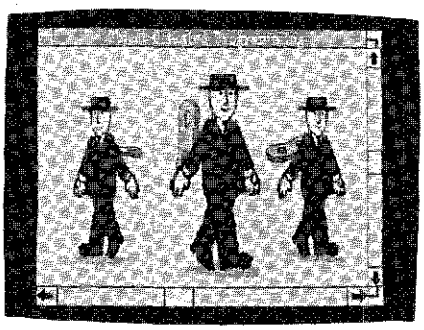
What's important here is that windows are objects like everything else in the language. You make as many as you need, and they do what they're supposed to, mostly on their own.

So it's natural and easy to put windows in your application programs. And users will benefit from Windows when your programs run, as much as you do when you write them.

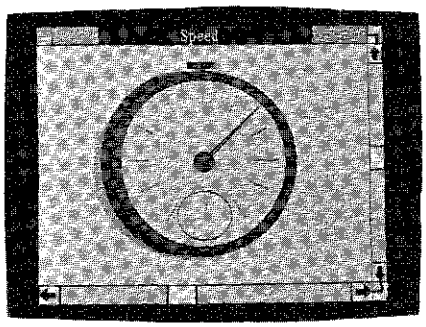
To order, just send in the coupon. Or call **312-491-2370** for more information. You'll see that ACTOR will do Windows, and more.



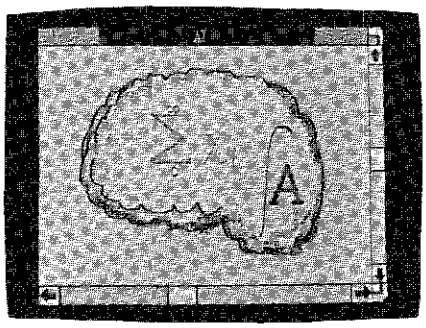
ACTOR™ is a new, interactive programming system, the first with Microsoft® Windows. You have Windows when you write a program, and users have Windows when they run it.



Moreover, ACTOR is a new, object-oriented programming language. Winding objects up and turning them loose is a lot easier, more productive, and more fun than old fashioned programming.



Thanks to a new method of "incremental garbage collection," ACTOR never has to slow down to clean up memory like other advanced languages. You can even use it for real-time control. It's that fast.



ACTOR offers all the features of an ideal artificial intelligence language, but in a familiar, Pascal-style syntax. Which makes artificial intelligence programming easier. And what could be smarter than that?

Please send me more information about ACTOR.

Name_____

Address_____

City_____ State_____ Zip_____

Daytime phone_____

☐ Check or money order enclosed. ☐ Bill my VISA. ☐ Bill my MasterCard.

Card number_____ Exp. date_____

☐ I'd like to place an order while I'm at it. (ACTOR is not copy protected and has a 60-day money back guarantee. Educational pricing is available. ACTOR requires an IBM PC/XT or AT or equivalent. 640K and hard disk is recommended.)

☐ ___ copies of ACTOR with Windows runtime support at $495 each. $_____

☐ ___ copies of ACTOR, including Windows complete with MS-DOS Executive, at $550 each. $_____

■ Please add $5 for shipping within the U.S., $30 overseas. $_____

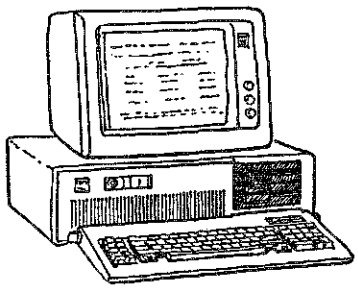
■ Total. $_____

The Whitewater Group®

Technology Innovation Center
906 University Place, Evanston, IL 60201

Circle Number 73 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



programmer's journal
The Resource Journal for IBM PC Programmers

Volume 4, Number 6 November/December 1986

CONTENTS

NOTE FROM THE EDITOR 2

LETTERS 3

PJ FORUM: DTACK GROUNDED, The Junk Mail Flyer *by Hal W. Hardenbergh* 7
The Clipper, disk access schemes, Word Perfect, HBASIC and strings

REDIRECTIONS: Unix Books, Part 1 *by Frank D. Greco* 15
A critical survey of the literature on Unix

REDIRECTIONS: How Many Amoebas at One O'clock? *by M.J. Budlong* 21
Ruminations on Artificial Intelligence

DOS: Parents, Children, Redirection, and Piping with DOS Functions 45H and 46H *by George Defenbaugh* 22
Two DOS functions explained

DOS: Terminate and Stay Resident *by M. Steven Baker* 26
Documented and undocumented TSR functions in DOS

PASCAL: Turbo Pascal Inline Code *by James M. Smith* 32
Putting assembly code in Pascal programs

PASCAL: Assembler Source Code for DOS Function Calls *by Steven K. O'Brien and Wing Kit Cheung* 39
An addition to Arvind Kumar's "Using DOS to Call Programs from Turbo Pascal"

C: Alignment Problems in C and Managing Memory Allocation *by William James Hunt* 40

C: The Proposed ANSI C Language Standard, Part 4 *by Rex Jaeschke* 44
The Draft Standard at last, ftell and fseek limitations, internationalization, and more on trigraphs

LASER PRINTERS: The Canon A1 Laser Printer *by Paul Barkley* 46
Coordinating the Canon A1, HP font cartridges, Word Perfect, and pitch, point size, proportional spacing, etc.

REVIEW: Genifer: A Powerful dBASE Applications Generator *by Paul Barkley* 54
What's a commercial applications generator like and why should you buy one?

REVIEW: Advanced Trace86: *by Peter Feldman* 58
A debugger and more

PJ BILLBOARD 63

HOUSEKEEPING 64
Advertiser Index, Peer Review

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

programmer's journal
The Resource Journal for IBM PC Programmers

Publisher:	Elizabeth Oakley
Editor:	Robert Keller
Managing Editor:	Erin O'Connor
Technical Editor:	M. Steven Baker
Production:	Ralph Bentley
Promotion:	Robert Tappan
Circulation:	Gary Hoyer
Cover:	Ralph Bentley
Production Assistance:	Betsy Wemple

Contributing Writers:

M. Steven Baker	Frank D. Greco
Paul Barkley	Hal W. Hardenbergh
M.J. Budlong	William J. Hunt
Wing Kit Cheung	Rex Jaeschke
George Defenbaugh	Stephen K. O'Brien
Peter Feldman	James M. Smith

Founder: Greg Estes

Business and Editorial Offices:	Programmer's Journal 29 West 29th Ave., Suite 5 Eugene, OR 97405 or P.O. Box 30160 Eugene, OR 97403 (503) 484-2162
Subscription Inquiries:	Programmer's Journal P.O. Box 3000 Dept. EE Denville, NJ 07834 (503) 484-2162

Subscriptions: Bi-monthly rates $24 U.S. ($30 in Canada and $40 elsewhere) 2 years $44 U.S. ($55 in Canada and $75 elsewhere) 3 years $64 U.S. ($80 in Canada and $110 elsewhere)

Articles and letters in Programmer's Journal represent the views of their authors and not necessarily those of the publisher. Letters to the editor become the property of Programmer's Journal and may be edited and published.

Advertising: For ad rates, deadlines, and placement, contact Ralph Bentley or Robin Tappan, (503) 484-2162, or write to Programmer's Journal, P.O. Box 30160, Eugene, OR 97403.

IBM and Personal Computer are trademarks of International Business Machines; dBASE II and dBASE III are trademarks of Ashton-Tate.

Programmer's Journal (ISSN 0747-5861) is published bi-monthly by Oakley Publishing Company. Application to mail at second-class rates is pending at Eugene, OR.

POSTMASTER: Send address changes to Programmer's Journal, P.O. Box 3000, Dept EE., Denville, NJ 07834.

© 1986

NOTE FROM THE EDITOR

386

We took a look at that Compaq 386 that no one seems to be able to make up their minds about down at our local computer store. It turned in 3800 Dhrystones using register variables and 232K Lahey SWhetstones.

Peer Review 4.4

William Hunt's article on writing memory resident programs in C gathered the most Peer Review votes in issue 4.4, followed by Hal Hardenburgh's PJ Forum, which was about tomorrow's merchant microprocessors and various other topics. In a tie for third was Graham Pearson's review of DOS function calls, Tom Swan's article on passing Turbo procedures as parameters, Michael Abrash's optimizing for speed, and Miriam Liskin's dBASE article. We'll start saving our pennies for a BMW for the winner.

Newsstand Sales

Speaking of pennies, you newsstand readers, think about subscribing to *PJ*. We only break even on rack sales.

Babies

Liz, our publisher, is about to have her third child, and so is our production manager, Ralph. Your subscription to *PJ* will not only bring the magazine to your door, it will pay for a week of diapers. In addition, the first two people who subscribe may, just may, have the children named after them.

—Robert □

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Saving and Restoring the Stack in BASIC

I am writing in regard to "PUSHING and STACKING" in the LETTERS column in Vol. 3, No. 1.

Chris Rivers writes about traps when relying on the size of the BASIC stack when calling routines. I recognize the problem because it has bitten me several times. At least I now know why. What I do not know is how to do what Chris says to do in the last paragraph: Set my own stack and then restore BASIC's stack before returning.

Can you be of any help?

Edward T. Grogan
Columbia, MD 21044

Steve Baker replies:

The CALL statement is normally used for calling machine language programs from IBM or GWBASIC. The syntax of the call statement is as follows:

```
CALL numvar (variable1,variable2,...)
```

where numvar is the offset into the current segment and variable1, etc., are a list of optional variables. The current segment is either the default (BASIC's segment) or the segment defined by the last DEF SEG statement.

As I recall, PC DOS BASIC guarantees that any routine called with the CALL statement has at least a 16-byte stack on entry. In most programs, unless a lot of BASIC data space has been allocated, there is usually much more stack space than this available. The assembly language fragment below can be used to save BASIC's stack on entry and restore it on exit. The called routine must preserve the segment registers DS, ES, SS, and BP, and the stack must be cleaned of parameters on exit.

If parameters are passed from the BASIC program, they will be placed on stack.

```
bas_entry:
        push    bp              ;save BP if needed
        mov     bp,sp           ;only needed when parameters on stack
;
; get any arguments off stack into registers or memory here if needed

        mov     cs:oldss,ss     ;and save entry SSEG
        mov     cs:oldsp,sp     ;and stack Pointer
;
        mov     ax,cs           ;set AX=CS
        mov     ss,ax           ;now setup local stack
        mov     sp,offset newstk
;
;  save any other segment registers which will be used
;
        call    routine         ; either call your routine
                                ; our put code INLINE
;
        mov     ss,cs:oldss     ; restore SSEG and SP
        mov     sp,cs:oldsp
        pop     bp
;
        ret                     ;if NO parameters on stack

; if parameters were placed on the stack by BASIC, then they must be
; removed by your routine

        ret     n               ;n = 2 x number of parameters on stack


oldsp   dw      0               ;space for saving entering stack
oldss   dw      0

        db      16 dup(?)       ;stack

newstk  dw      0
```

M. Steven Baker
Eugene, OR 97401

More on New Trick, Questions for Borland

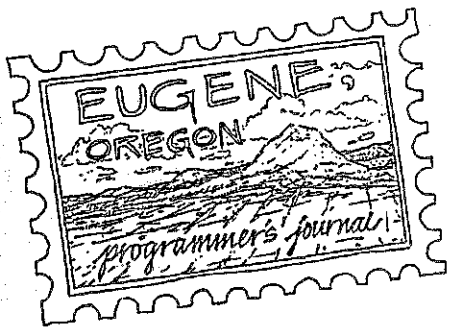
I have some suggestions in regard to details in "Teach an Old Compiler a New Trick" by Tom Swan in the July/August 1986, issue of *Programmer's Journal*.

First, I'd like to suggest a somewhat more efficient way to write the CALL Procedure:

```
                Original

PROCEDURE call [process:proc];
BEGIN
  Inline($89/$EC        (* mov sp,bp   ; reset stack and *)
         $5D/           (* pop bp      ;bp registers     *)
         $58/           (* pop ax      ; return address  *)
         $5B/           (* pop bx      ; process address *)
         $50/           (* push ax     ; reverse the two *)
         $53/           (* push bx     ;addresses        *)
         $C3/           (* ret         ; execute process *)
        );
END; (* call *)

                Modified

PROCEDURE call(process:proc);
BEGIN
  Inline($89/$EC/       (* mov sp,bp   ; reset stack and *)
         $5D/           (* pop bp      ;bp registers     *)
         $58/           (* pop ax      ; return address  *)
         $5B/           (* pop bx      ; process address *)
         $50/           (* push ax     ; save return adrs*)
         $FF/$E3        (* jmp bx      ; jmp to process  *)
        );
END; (* call *)
```

Clearly the modified version must execute a little bit faster; the process address is jumped to directly instead of pushed onto and then popped off (by the RET) the stack. I wouldn't have felt obliged to write about this had it not been for Michael Abrash's article on optimization for speed in the same issue—everyone should try to ensure that his or her code is as efficient as possible.

Secondly, Tom stated that you cannot pass Procedures with parameter lists, nor can you pass Functions. I didn't think that you shouldn't be able to do either of these, so I simply tried matching the Call Procedure of Function to the Procedures or Functions that it wants to be able to call. Well, that worked the first time out. The following little sample of code uses a Function with parameters to be accessed through Call. The only rules that such Procedures or Functions must follow is



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

that the Procedure or Function address must be the last parameter of the Call Procedure or Function and the parameter list of the Call Procedure or Function must be the same as that of the Procedures or Functions that it wishes to call (excluding the final parameter of the Call parameter list, the Procedure or Function address—and that a Call Procedure cannot call a Function and vice versa).

```
PROGRAM TestCal2;          (* Test passing function parameters of
                              functions with parameter lists *)

TYPE
   Proc = integer;           (* Procedure address *)

VAR
   q : real;

FUNCTION Call(i:integer; Process:PROC):real;
BEGIN
   InLine($89/$EC/      (* mov sp, bp    ; reset stack and      *)
          $5D/          (* pop bp        ;bp registers         *)
          $58/          (* pop ax        ; return address      *)
          $5B/          (* pop bx        ; address of function
                                          indirectly called     *)
          $50/          (* push ax       ; put return address
                                          back on stack         *)
          $FF/$E3       (* jmp bx        ; jump to indirectly
                                          called function       *)
         );
END;

FUNCTION r(i:integer):real;
BEGIN
   writeln('passed integer = ',i);
   r:=i;
END;

BEGIN (* main program *)
   q:=call(12,ofs(r));  (* note that don't need p:=ofs(r), though
   can use it *)
   writeln('q=',q);
END.
```

That's all I've got on this subject. However, since I've been working with Turbo Pascal for some time now, I've been wondering about a few things that it does (actually, I'm hoping that you can get an answer from Borland on the subject). At the beginning and end of every Procedure and Function, after saving or restoring BP, there is a JMP to the following statement. This seems terribly inefficient, especially with the 8088, since the 8088 does not execute JMPs terribly quickly and these JMPs clear the prefetch queue (which is the only reason that I can see for doing it). There doesn't seem to be any reason to clear the queue at that point. Does anyone know any reason for this code? The other question (again to Borland) is: Do they ever plan to use the 80186 extensions? These would result in faster and smaller code. Since the V20 and V30 chips are available as direct replacements of the 8088 and 8086 (the only chips in the 16-bit Intel family that don't have the 80186 extensions), it would be possible to offer a Turbo Pascal compiler that would run on everything but the 8088 and 8086—since the V20 and V30 could be swapped for these chips. Of course, this would make the compiler (and the code that it produces) a little less general than it currently is, so I could see why Borland might not want to commit to the extensions completely (but couldn't they have a TURBO186 version of the compiler so that those who had the better processors could take advantage of them?).

Roderick K. Graves
Sunnyvale, CA 94086

Tom Swan replies:
Mr. Graves, your improvement to PROCEDURE Call is slightly more efficient than my original routine, as you suggest. (Both procedures are the same size, 8 bytes.) In a test loop of 100,000 iterations, I clock the original at 13.99 seconds and the new version at 13.57 seconds—about 3% better for yours.

I agree also with your comments on calling functions and routines with parameters, but I'm concerned that your solution requires a different Call procedure for each function type and for each parameter list arrangement. I caution readers to weigh carefully the advantage of a parameterized Call against this added complexity. Still, your ideas do seem to work.

On your questions for Borland, I suggest calling their technical support staff. They are a very helpful and eager bunch. And please let us know what you find out.

Tom Swan
Lititz, PA 17543

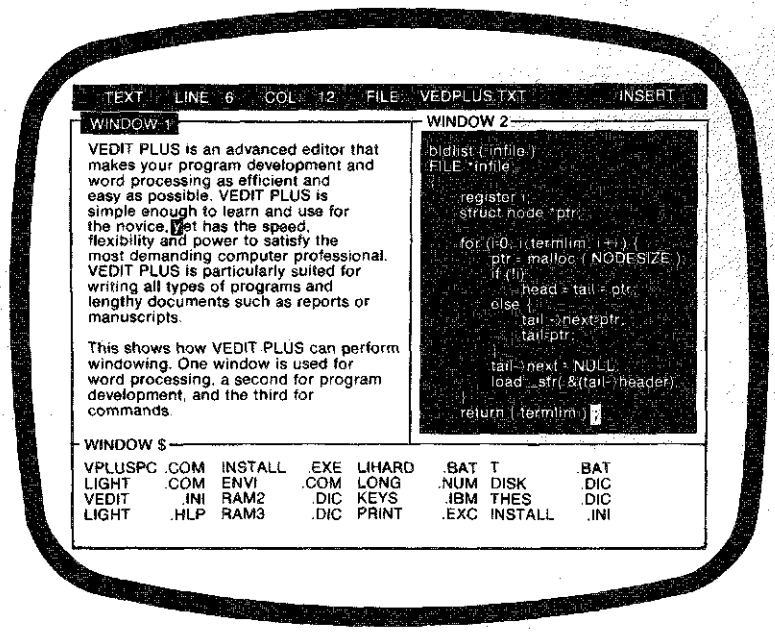
Modula-2 Missing

I've enjoyed my subscription to *PJ* very much. I'm probably responsible for dozens of other subscriptions since I have a growing client base to whom I repeatedly recommend it.

However, I recently became enchanted with Modula-2 and have decided it will be the language of the future (my future anyway). This wasn't an easy decision for a former C-bigot. I really don't miss C as much as I thought I would, but I do miss reading publications like *PJ* and finding articles that are informative about the language I'm using. Quite frankly, I am at a total loss to explain how a publication so right-on-the-mark in so many other ways could have missed the most significant contribution to computer languages in a decade. If Pascal and dBASE have caught your attention, how could Modula-2 have escaped it? Please consider expanding your publication to include articles on this language. I'm sure you won't be sorry.

Dale McKinnon
Salt Lake City, UT 84147 □

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED


TEXT LINE 6 COL 12 FILE: VEDPLUS.TXT INSERT
WINDOW 1
VEDIT PLUS is an advanced editor that makes your program development and word processing as efficient and easy as possible. VEDIT PLUS is simple enough to learn and use for the novice, yet has the speed, flexibility and power to satisfy the most demanding computer professional. VEDIT PLUS is particularly suited for writing all types of programs and lengthy documents such as reports or manuscripts.
This shows how VEDIT PLUS can perform windowing. One window is used for word processing, a second for program development, and the third for commands.
WINDOW 2
bldlist (infile)
FILE *infile;
register i;
struct node *ptr;
for (i=0; i<termlim; i++) {
ptr = malloc (NODESIZE);
if (!i)
head = tail = ptr;
else {
tail->next=ptr;
tail=ptr;
}
tail->next = NULL;
load_str(&(tail->header)
}
return (termlim);
WINDOW $
VPLUSPC .COM INSTALL .EXE LIHARD .BAT T .BAT
LIGHT .COM ENVI .COM LONG .NUM DISK .DIC
VEDIT .INI RAM2 .DIC KEYS .IBM THES .DIC
LIGHT .HLP RAM3 .DIC PRINT .EXC INSTALL .INI

VEDIT PLUS

#1 CHOICE IN PROGRAMMABLE EDITORS

VEDIT PLUS has been the #1 choice of professionals since 1980. Our latest release is even better - you can open windows to simultaneously edit several files, access many editing functions with pop-up menus, use keystroke macros to speed editing, and run other programs or DOS commands from within VEDIT PLUS.

Whether your needs are program development, technical writing or word processing, VEDIT PLUS is your answer. With over 40,000 users you can depend on VEDIT PLUS to perform consistently and reliably. It is simple enough to learn for the novice, yet has the speed, flexibility and power to satisfy the most demanding professional.

Compare. No other editor is as powerful - unlimited keystroke macros, instant 'off-the-cuff' command macros utilizing a complete programming language, single command file comparison, special word processing and programming features. No other editor is as easy to use - on-line help, pop-up menus, 75 page tutorial, 380 page manual, and VEDIT PLUS is completely customizable.

Fully supports color windows on IBM CGA & EGA, and even windows on most CRT terminals (including CRT's connected to an IBM PC). Available for IBM PC, TI Professional, Tandy 2000, DEC Rainbow, Wang PC, MS-DOS, CP/M-86 and CP/M-80. Order direct or from your dealer. $185

"To sum things up, VEDIT PLUS is a small, fast, sophisticated editor with a wealth of features and a good macro language. It offers many rewards for the dedicated programmer."
Computer Language, Chris Wolf, Scott Lewis, Mark Gayman 6/86

"VEDIT PLUS is a wholly remarkable program: blindingly fast, extremely powerful, and highly flexible."
Profiles Magazine, Robert Lavenda 4/86

VEDIT PLUS FEATURES

- Simultaneously edit up to 37 files of unlimited size.
- Split the screen into variable sized windows.
- 'Virtual' disk buffering simplifies editing of large files.
- Memory management supports up to 640K.
- Execute DOS commands or other programs.
- MS-DOS pathname and CP/M user number support.
- Horizontal scrolling - edit long lines.
- Flexible 'cut and paste' with 36 text registers.
- Customization - determine your own keyboard layout, create your own editing functions, support any screen size, any CRT.
- Optimized for IBM PC/XT/AT. Also 132 column & up to 70 lines.

EASY TO USE

- Interactive on-line help is user changeable and expandable.
- On-line integer calculator (also algebraic expressions).
- Single key search and global or selective replace.
- Pop-up menus for easy access to many editing functions.
- Keystroke macros speed editing, 'hot keys' for menu functions.

FOR PROGRAMMERS

- Automatic Indent/Undent for 'C', PL/I or PASCAL.
- Match/check nested parentheses, i.e. '{' and '}' for 'C'.
- Automatic conversion to upper case for assembly language labels, opcodes, operands with comments unchanged.
- Optional 8080 to 8086 source code translator.

FOR WRITERS

- Word Wrap and paragraph formatting at adjustable margins.
- Right margin justification.
- Support foreign, graphic and special characters.
- Convert WordStar and mainframe files.
- Print any portion of file; separate printer margins.

MACRO PROGRAMMING LANGUAGE

- 'If-then-else', looping, testing, branching, user prompts keyboard input, 17 bit algebraic expressions, variables.
- CRT emulation within windows, Forms entry.
- Simplifies complex text processing, formatting, conversions and translations.
- Complete TECO capability.
- Free macros: • Full screen file compare/merge • Sort mailing lists • Print Formatter • Main menu

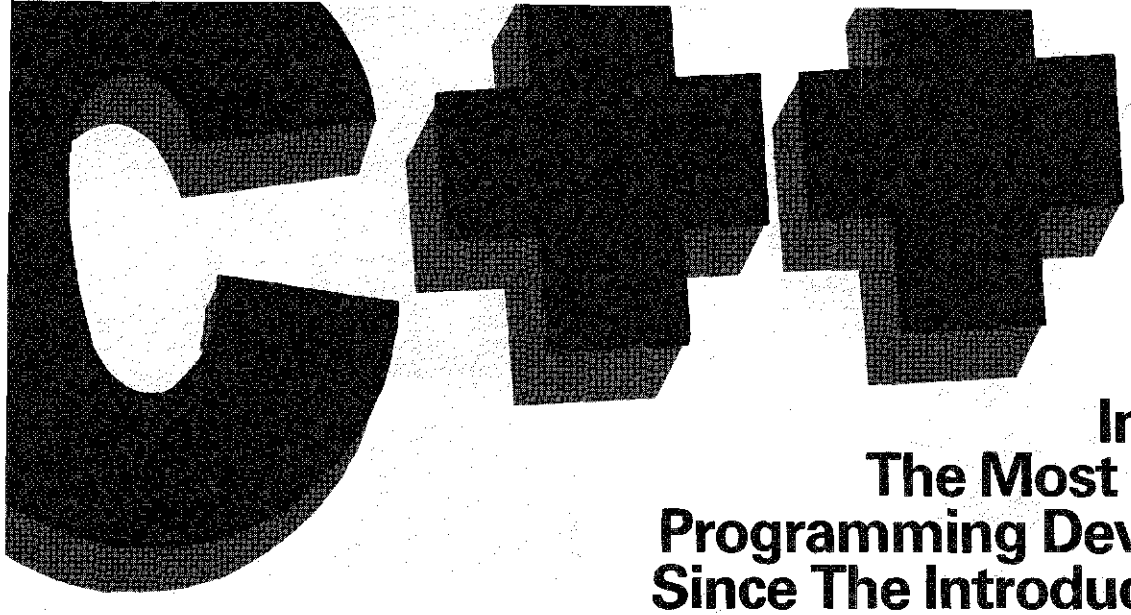
VEDIT and CompuView are registered trademarks of CompuView Products, Inc. MS-DOS is a registered trademark of Microsoft. CP/M is a registered trademark of Digital Research. WordStar is a registered trademark of MicroPro.

Circle Number 56 on the Reader Service Card

CompuView

1955 Pauline Blvd., Ann Arbor, MI 48103 (313) 996-1299, TELEX 701821

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



Introducing The Most Important Programming Development Since The Introduction Of C:

ADVANTAGE C++

Exclusively From LIFEBOAT For PC/MS-DOS

Finally there's a programming language that enhances C, instead of making it obsolete! **ADVANTAGE C++**, developed by AT&T, is a major programming breakthrough. By introducing the concept of classes, it enables C programmers to use object-oriented programming methods. **ADVANTAGE C++** gives you greater efficiency, flexibility and reliability than ever before—and allows you to more productively build large and sophisticated applications.

ADVANTAGE C++ All The Benefits Of C Without Its Limitations!

- Opens the door to object-oriented programming.
- Allows programs with greater resilience and fewer bugs.
- Lets you write reliable, reusable code that is easier to understand.
- Has many enhancements over C, yet maintains full compatibility with existing C programs.

Advantages Only ADVANTAGE C++ Can Give You:

- Operator Overloading—Allows simple, reliable user-designed types.
- Function Name Overloading—Simplifies function names and argument lists.
- Guaranteed Programmed Initialization—Ensures automatic initialization of all data objects before their use.
- Guaranteed Programmed Type Conversion—Ensures consistent conversions from one user-designed type to another.
- Optional Strong Type Checking—Weeds out type mismatches at compile time.
- Classes—Similar to structures; provide syntax for user-designed data types and encapsulation of access functions with data objects.
- Data Abstraction—Makes code easily reusable and more resilient.
- Data Hiding—Improves software reliability.
- Inheritance—Enables generic code written for more abstract types such as 'list' or 'windows' to be used by more specific types.
- Constant Data Types—Prevent inadvertent alteration of fixed values, such as hardware addresses.
- Reference Data Types—Improve the efficiency of argument passing.
- Inline Functions—Remove the overhead of calling external functions.
- Heap Management—Simplifies the use of dynamic memory.

Why be limited to just C...when you can have all these pluses! **ADVANTAGE C++** includes libraries for stream I/O and complex math—and versions are now available for Lattice C and Microsoft C. Call today to order or to obtain a complete technical specification sheet:

1-800-847-7078
In NY: 914-332-1875

LIFEBOAT

The Full-Service Source for Programming Software.

Circle Number 81 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

DTACK GROUNDED, The Junk Mail Flyer

by Hal W. Hardenbergh

Quickies:
A rare and unaccountable outbreak of intelligence and rationality has suddenly struck *InfoWorld*. In two consecutive issues there were three articles, including a Kee opinion column, pointing out that 95% of the PC owner/users needed multitasking like they needed AIDS. And that there was fierce resistance to MS DOS Versions 4.0 and 5.0, which are now in beta-site evaluation. MS DOS 4.0 and 5.0 require that all those programs written for MS DOS 2.x and 3.x be rewritten. For some reason, a lot of software companies don't want to do that for a lousy 5% of the market. And when the absence of that software is noted, that market is gonna be a lot smaller than 5%.

This rare and unaccountable outbreak of intelligence and rationality has not yet infected the staff of *PC Magazine*. For instance, Jim Seymour has just informed *PC*'s readers that the real importance of the Intel 386 is that the mass personal computer marketplace can now have a virtual-memory computer! That makes Jim the leading candidate for the 1986 Jeanyates Award.

IBM has this FORTRAN compiler which it licenses to its mainframe sites for $12,500 per month. That's $450,000 over a three-year period. Rumor has it that there is a heated internal dispute over whether to adapt this same compiler for the PC and "license" it for a one-time fee of about $200. I don't make these things up, folks!

Sleeper Compartment:
A couple of flyers back I told you I thought the Fairchild Clipper just might be a sleeper. I don't think it will appear in a KUJ $995 rock-shooting toy in the foreseeable future, but that is not its intended market. Its intended market is the upcoming 32-bit engineering workstation market, where it will buck heads with the MicroVax II, the IBM RT, and the merchant-micro machines based on the 68020 and the 80386. The Clipper can outperform the 68020 and the 80386 by about a factor of 2, the MicroVax II by a factor of 4, and the IBM RT by a factor of 5 or more. And its price of $1500 per 3-chip board is not too steep for a machine which bucks heads with $17–$27K machines from other vendors.

Intergraph has just become the first company to introduce an engineering workstation based on the Clipper. It will not be the last. For $25K you get an 80-megabyte disk, 6 megabytes of RAM (50% more than I have on my DTACK/IBM attached processor), a mega-pixel non-interlaced color display with a now-standard color palette, and some other stuff. A keyboard, for instance. And Intergraph is big enough to back that machine up with marketing and software. I loved the two-full-page ad in my latest *Electronic News*.

I think the $27K Sun Microsystems 68020-based engineering workstations might be in trouble. I know the IBM RT is in trouble, but then it already was. IBM itself crippled the RT as sold by folks other than IBM's own sales force by subtracting all the useful software and also subtracting its 5080 mega-pixel display. As a result of this foolishness, the RT's sales are right up there with the UNIX PC's. And that was before the Intergraph announcement.

So let's discuss some more reasons why I like the Clipper and think the 68020 and the 80386 are going to have rough going competing with it. The secret is its twin cache. Let me quote from Fairchild marketing director Tom Miller in *Electronic Engineering Times*.

Single-user systems, running a single task at a time, can do well with a simple direct-mapped one-way associative cache system like that used on many previous-generation workstations. But as DEC realized when it went to the VAX architecture, this simple caching strategy can cripple applications running under a multi-tasking operating system like UNIX. They cause what's called cache-thrashing—senseless filling and refilling.

". . .Clipper uses separate caches for instructions and data, each tightly coupled to the CPU, organized two-way set associative, and including both write-through and copy-back caching strategies." Translation: Howard Sachs is one smart fellow when he's designing CPUs instead of suggesting that 1-2-3 be rewritten in C.

The Next Boring Subject:
The next subject which I am going to hammer on until all of you get sick and tired of it is the desperate need in these days of 16- to 33MHz CPU clock rates for an efficient CPU-to-multi-megabyte DRAM memory interface.

There are two ways to achieve a better interface with DRAM: large, properly-designed caches—one for instructions and one for data. Clipper has this. The other way is built-in logic to take advantage of nibble mode, which provides (typically) a 33% speed increase when reading. And reads are about six or seven times more common than writes. Clipper has built-in logic to take advantage of nibble mode. And that is why the Clipper is about twice as fast as a 68020 or 80386. Without the caches and the nibble DRAM access mode, the 33MHz clock rate would be useless because of the infamous von Neumann bottleneck—the CPU to memory bus-bandwidth.

If either the 020 or the 386 is going to try to knock heads with the Clipper in the $17K-$27K bracket, then it is most certainly going to have to have large, dual caches which are two-way set associative. Translation: $$$$$$$$$.

Here is where the $1500 Clipper has the $250 020 and 386 whipped: Its caches and ancillary logic have been reduced to two identical VLSI chips having about a quarter-million device-equivalents each. So each of those cache chips is about as complex as an 020 or a 386! And, being VLSI—the production cost is going to be a hell of a lot less than a cache built up of discrete chips! In other words, the Clipper in its intended marketplace is more cost-effective at $1500 than the 020 or

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

386 is at $250. That fact is so important that it is almost, but not quite, irrelevant that the Clipper has built-in nibble-mode logic and the 020 and 386 do not.

I learned a long time ago that in the electronics business *one* of something costs a million dollars, but a *million* of something costs a buck apiece. It will be impossible for workstation vendors who each have to design their own caches and build them with discrete chips to compete against the mass-production, already-designed VLSI cache chips which come as standard equipment on that $1500 CPU board.

What to Do?

Several years back Intel's Andy Grove publicly questioned what the heck the chip vendors were going to do with a million transistors per chip when that level of integration became available. Besides make megabit DRAMs, that is. One of the answers is that it will become possible to package a complete 32-bit computer architecture on a single chip. The CPU, dual caches, a double-precision floating point logic unit, and full memory management. The industry is not quite at that point yet, which is why the Clipper comes as a 3-chip set with a total of about three-quarters of a million transistor-equivalents (the Clipper is not quite a complete real-world computer architecture).

Both the 68020 and the 386 use roughly a third of a million transistor-equivalents, so neither can even come close to being complete computers as a single chip. Both use external floating point units; the 68881 is in production, and the 387 is a piece of paper. Very nice paper. But a third of a million transistor-equivalents is more than one needs for just a CPU; what shall we do with the remaining available logic?

Intel, following the policy it established with the 80286, made the decision to spend the remaining silicon on internal memory management, which is highly desirable in a multi-user/multi-tasking environment. The mass personal computer market is a single-user/single-tasking environment. How many PC ATs ever run in the memory-protected mode? As many as 17? Memory management is as useless as tits on a boar hog in the mass personal computer market. If you say otherwise, you're describing a business computer!

Motorola made the decision to go with an internal instruction cache. I don't think they did that with us personal-computer users in mind, but the result is that the 020 is better for us—you and me—than the 80386.

I wonder if IBM will attempt to introduce its 386-based PC as another multi-user machine like the AT was intended to be, or whether they have wised up that folks buy personal computers for personal use?

Caution Dept:

The Clipper is great hardware and highly competitive in its intended niche. . . as hardware. There is about as much software available for Clipper as there was for the 68000 in 1980. For that reason, the companies looking for early success with Clipper had better have sizeable resources. The MicroVax II and Sun 68020 machines are not going to disappear overnight!

Comparing the performance of the 68020 and the 386 is a complex matter. The correct answer to "Which is faster?" is "Both of them!" A comparison has to be based on a highly specific configuration.

If unlimited funds are thrown at those two CPUs to gussy them up, the 386 comes out ahead because of two cycles per external cache memory fetch and because Intel has commissioned Weitek to develop an interface chip which will tie Weitek's double-precision math chips to the 386. Such a configuration would be a lot faster than a 68020/68881 with an external cache! But the price tag would be so heavy that I just might want to look at the far-cheaper $1500 Clipper at that point. . .

Beaucoup Blue Sky:

The high density floppies we now have and which I have been using for over 6 months are almost so large and so fast that I almost don't want anything larger or faster. Almost. I have been tentatively looking at hard disk specifications of late. A brief review:

Your vanilla real-computer floppy disk, the one on the IBM PC, holds 360K formatted. It rotates at 300 rpm and transfers data at 250Kbits/sec.

The HD floppy we are using, which has been in common use in Japan for about 3 years now, is essentially the same as the one in the PC AT. Which, by the way, nobody seems to use. . . in the AT. It holds 1.2 megs the way IBM formats it and 1.28 megs the way we format it. It rotates at 360 rpm and transfers data at 500Kbits/second.

Vanilla real-world floppies are almost never used in an efficient manner, so the average throughput when reading large files is on the order of 10- to 12Kbytes/sec. Us folks at Digital Acoustics are very fond of using hardware efficiently, so we get an average throughput of 40Kbytes/sec from our HD floppy when reading large files.

The typical small-computer type Winchester rotates at 3600 rpm, exactly ten times as fast as our HD floppy. It transfers data at 5Mbits/sec, exactly ten times as fast as our HD floppy. So a DMA-equipped 68000-based computer, with a 16-bit DMA port, can naturally read large files at an average throughput of 400Kbytes/sec? HO HO HO!

With a rotation every 16.7 msec instead of 167 msec, the track-to-track stepping and settling time becomes very important. There is about a 5–1 range in performance in this parameter in Winchesters. There are 3 types of head positioning mechanisms, and you gets what you pays for. But more important than that, for some reason hardly anybody reads a track in sequence. A specific example:

Digital Acoustics has a Haba disk for the Atari ST, an early version which we bought last August. It works. As I reported 'way back then, it loaded consecutive 30K files at a rate of almost 2 per second for an average throughput of 56K/second. Well, that included the seek time and probably the catalog read time. James took off the cover and fired up an oscilloscope, and this is what he found:

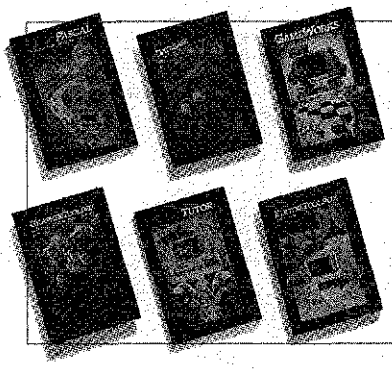
The Haba disk reads a 512-byte sector in .765 msec. Let's call that ¾ of a millisecond, O.K.? Then it does nothing for 3 milliseconds, which is 4 sector times. Then it reads another sector. In other words, it reads every fifth sector. So instead of being able to read an 8K track in 16.7 msec (one rotation), it reads that track in 83.3 msec (five rotations). So if we ignore the track-to-track stepping and settling time, the Haba hard disk has a maximum throughput of only 96Kbytes/sec instead of 480Kbytes/sec. Neither of those rates can actually be achieved; how close one comes depends on how much money one spends on the head-positioning mechanism.

But since I already have a (floppy) disk which has an actual throughput of 40Kbytes/sec and which I can back up in 66 seconds, if I am going to put my precious data on a hard-to-back-up Winchester, I want to be trying to get as close to

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Why more than half a million people are using Turbo Pascal

Because Turbo Pascal is faster than any other Pascal compiler—it's become the industry standard. And because Turbo Pascal is backed by a complete range of "tool boxes" that give you most of the programming tools you'll ever need.



- Turbo Pascal® 3.0 with 8087 and BCD
- Turbo Database Toolbox™
- Turbo Graphix Toolbox®
- Turbo Tutor® 2.0
- Turbo Editor Toolbox™
- Turbo GameWorks®

The Turbo Pascal family is never static, but is continuously expanding, with new products like Turbo Editor Toolbox and Turbo Gameworks.

> "Language deal of the century . . . Turbo Pascal"
> *Jeff Duntemann, PC Magazine*
>
> "Turbo Pascal has got to be the best value in languages on the market today"
> *Jerry Pournelle, BYTE Magazine*
>
> "This compiler, produced by Borland International, is one of the best programming tools presently available for the PC."
> *Michael Covington, PC Tech Journal*

The secret of software success is not merely low price, but top quality allied with complete documentation, like our 400-page reference manual.

All of which are some of the reasons why Turbo Pascal is clearly the leader, and the recipient of awards like *PC Week's* **"Product of the Year"** and *PC Magazine's* **"Award for Technical Excellence."**

NEW! *TNT* A DYNAMITE DEAL!*

Turbo 'N Tutor

Turbo Pascal *and* Turbo Tutor for only $125.00. You save nearly $15.00, and have everything you need to get up and go with Turbo Pascal!

A whole family of tools

Success breeds success, so the Turbo Pascal family has flourished. Your choices now include:

☐ Turbo Pascal 3.0 combines the fastest Pascal compiler with an integrated development environment, *and* 8087 math co-processor support *and* Binary Coded Decimals.

☐ Turbo Database Toolbox a perfect complement to Turbo Pascal. It includes a complete library of Pascal procedures that allows you to search and sort data and build powerful database applications.

☐ Turbo Graphix Toolbox includes a library of graphics routines for Turbo Pascal programs. Lets even beginning programmers create high-resolution graphics with an IBM,® Hercules,™ or compatible graphics adapter. Does complex business graphics, easy windowing, and stores screen images to memory.

☐ Turbo Tutor 2.0 teaches you step-by-step how to use Turbo Pascal, with commented source code for all program examples on diskette.

An offer you can't refuse

You can save $119.75 when you choose the Turbo Jumbo Pack. Six different Turbo Pascal products for only $299.95.

You get Turbo Pascal 3.0 *and* Turbo Editor Toolbox *and* Turbo Tutor 2.0 *and* Turbo Graphix Toolbox *and* Turbo GameWorks *and* Turbo Database Toolbox!

So act now—rush to your dealer or call us today!

NEW! *Amazing value! Turbo Editor Toolbox includes MicroStar, a full-blown editor that also does windows!* You get ready-to-compile source code and a 200-page manual that tells you how to integrate the editor procedures and functions into your programs.

Also includes MicroStar, a complete editor with full windowing capabilities. (You could pay $100.00 or more for a program like MicroStar, but you get it free as part of our Turbo Editor Toolbox.) You can also use Turbo Editor (which of course integrates with Turbo Lightning™) to build your own word processor!

NEW! *Turbo GameWorks gives you the games you can play, write, rewrite, bend and amend!* Turbo GameWorks reveals the secrets and the strategies of game design. You're given source code, a 200-page manual, and the insight needed to write and customize your own irresistible games.

Also includes ready-to-play Chess, Bridge, and Go-Moku—an ancient Japanese game that can absorb you in reality for hours on end.

YES! I want the best!

To order by phone, or for a dealer nearest you,
call (800) 255-8008
in CA call (800) 742-1133

Copies	Product	Price	Totals
___	Turbo Pascal 3.0 w/8087 & BCD**	$ 99.95	$ ___
___	Turbo Pascal for CP/M-80	69.95	$ ___
___	Turbo Database Toolbox	69.95	$ ___
___	Turbo Graphix Toolbox*	69.95	$ ___
___	Turbo Tutor 2.0	39.95	$ ___
___	Turbo Editor Toolbox*	69.95	$ ___
___	Turbo GameWorks*	69.95	$ ___
___	Turbo Pascal & Turbo Tutor	125.00	$ ___
___	Turbo Jumbo Pack*	299.95	$ ___
	Outside USA add $10 per copy		
	CA and MA res. add sales tax		$ ___
	Amount enclosed		$ ___

Prices include shipping to all US cities.

Carefully describe your computer system:

Mine is: ___ 8-bit ___ 16-bit

I use: ___ PC-DOS ___ MS-DOS ___ CP/M-80 ___ CP/M-86

My computer's name and model is:

The disk size I use is: ☐ 3½" ☐ 5¼" ☐ 8"

Payment: VISA MC Money Order Check

Credit card expiration date ___/___

Card #

Name:

Shipping Address:

City:

State: Zip:

Telephone:

CODs and purchase orders WILL NOT be accepted by Borland. Outside USA make payment by bank draft, payable in US dollars drawn on a US bank.

NOT COPY PROTECTED
60-DAY MONEY-BACK GUARANTEE

If within 60 days of purchase this product does not perform in accordance with our claims, call our customer service department and we will gladly arrange a refund.

All prices are suggested list prices and are subject to change without notice.

Minimum System Requirements:
Turbo GameWorks, Turbo Graphix Toolbox, Turbo Tutor, & Turbo Editor Toolbox—192K. All other products, 128K.

*IBM PC, PCjr, AT, XT, and true compatibles.
**16-bit only.

TP 36

BORLAND INTERNATIONAL

Vive la différence

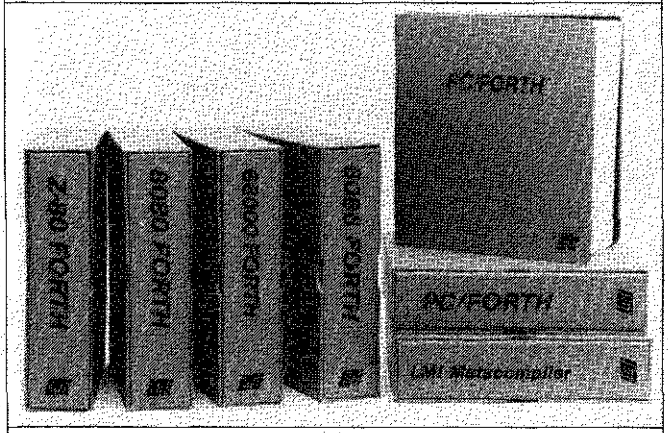
4585 SCOTTS VALLEY DRIVE
SCOTTS VALLEY, CA 95066
(408) 438-8400 TELEX: 172373

Borland products include Turbo Prolog; Turbo Pascal; Turbo Pascal for the Mac; Turbo Tutor; Turbo Editor Toolbox; Turbo Database Toolbox; Turbo Graphix Toolbox; Turbo GameWorks; Turbo Lightning; Lightning Word Wizard; Reflex: The Analyst; Reflex for the Mac; Reflex Workshop; SideKick; SideKick for the Mac; Traveling SideKick; and SuperKey—all of which are trademarks or registered trademarks of Borland International, Inc. or Borland/Analytica, Inc. Traveling SideKick is not in any way associated with Traveling Software, Inc. of Seattle, Washington.
CP/M-80 is a registered trademark of Digital Research, Inc. MS-DOS is a registered trademark of Microsoft Corp. IBM is a registered trademark of International Business Machines Corp. Hercules is a trademark of Hercules Computer Technology.
Copyright 1986 Borland International. BI-1082

Circle Number 3 on the Reader Service Card.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

TOTAL CONTROL with *LMI FORTH*™



For Programming Professionals:

an expanding family of compatible, high-performance, Forth-83 Standard compilers for microcomputers

For Development: Interactive Forth-83 Interpreter/Compilers

- 16-bit and 32-bit implementations
- Full screen editor and assembler
- Uses standard operating system files
- 400 page manual written in plain English
- Options include software floating point, arithmetic coprocessor support, symbolic debugger, native code compilers, and graphics support

For Applications: Forth-83 Metacompiler

- Unique table-driven multi-pass Forth compiler
- Compiles compact ROMable or disk-based applications
- Excellent error handling
- Produces headerless code, compiles from intermediate states, and performs conditional compilation
- Cross-compiles to 8080, Z-80, 8086, 68000, 6502, 8051, 8096, 1802, and 6303
- No license fee or royalty for compiled applications

For Speed: CForth Application Compiler

- Translates "high-level" Forth into in-line, optimized machine code
- Can generate ROMable code

Support Services for registered users:

- Technical Assistance Hotline
- Periodic newsletters and low-cost updates
- Bulletin Board System

Call or write for detailed product information and prices. Consulting and Educational Services available by special arrangement.

Laboratory Microsystems Incorporated
Post Office Box 10430, Marina del Rey, CA 90295
Phone credit card orders to: ***(213) 306-7412***

Overseas Distributors.
Germany: Forth-Systeme Angelika Flesch, Titisee-Neustadt, 7651-1665
UK: System Science Ltd., London, 01-248 0962
France: Micro-Sigma S.A.R.L., Paris, (1) 42.65.95.16
Japan: Southern Pacific Ltd., Yokohama, 045-314-9514
Australia: Wave-onic Associates, Wilson, W.A., (09) 451-2946

Circle Number 63 on the Reader Service Card

480K/sec as I can, not a lousy 96K/sec.

There may be some hard disks out there which are like what I want. Quantum, for instance makes a hard disk-cum-controller which has a track buffer. I think it reads a track at a time. But when we called Quantum and asked about the average throughput, the kid they had on the phone didn't know anything but "5 megabits per second," a phrase which he parroted repeatedly. Well, we have the same problem: Nobody can afford to put anybody who knows anything on the phone line.

A Request:
Maybe some of you folk out there have some info about hard disks and controllers and track buffers. Stuff which is not proprietary, that is. If so, would you mind photocopying it and sending it to me, please?

Alternatives:
There are many ways to skin a cat. How to run faster than what we now have? Hmmm . . . parallel processing. Those HD floppies cost only a tad over $100 a copy. Four of them could read 160Kbytes per second, more than twice the throughput of the Haba hard disk. Eight of them would hold ten megabytes and could read 320Kbytes per second . . .

I have four megabytes of DRAM on my DTACK system right now. In about a year I will have ten megabytes. So I boot into RAM disk every morning (maybe from parallel floppies?) and do everything from RAM. 68000-based RAM disks are amazingly fast. Every hour or so when I get up to stretch and sip a cup of water or whatever I hit the "backup" button and the floppy is updated. Perhaps this idea should be investigated further.

One could read the latest product release which is going to cure this problem forever and sit back and wait for that product to be shipped tomorrow afternoon at 3 o'clock. In 1982 that product was Syquest's cheap Winchester with removable media. In 1983 it was the Sony ten-megabyte 3.5 inch floppy using vertical recording techniques. In 1984 both Amlyn (using Dysan's money) and Drivetec (using Kodak's money) actually produced pilot runs of multi-megabyte 5.25 inch floppy drives. Neither of them panned out. In January of 1985 3M announced its 5.25 inch erasable optical disk, the one which held 300 to 500 megabytes. And Sir Clive was going to ship his semiconductor disk-on-a-wafer. In 1986 Kodak is trying to resurrect Drivetec and Toshiba has announced a 4 megabyte 3.5 inch floppy with a 1Mbit/sec data transfer rate using vertical recording. And in 1980, '81, '82, '83, and '84 bubble memory was going to take over the world. All of those breakthroughs listed above came with very nice brochures. So far, none of them got shipped at 3 o'clock the next day.

Waiting for the latest breakthrough can take a while . . .

There is an extremely simple solution to the problem of a big, fast hard disk and a fast reliable backup. It's available now. It was available last year, and the year before that, and . . . what's that? You want to know what this extremely simple solution is? Well, it's spelled "$$$$$$$$."

For an appropriate sum of money one can get a very good, very fast hard disk and a very fast and reasonably reliable tape backup. But that "sum of money what am appropriate" is for business, not personal, budgets. What I'm looking for is a faster disk which one can easily back up and which is priced appropriately for a personal computer.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

The Old Paradigm:
Paradigm is a rather unlovely word which is commonly used in technical writings when the author wants to obscure the fact that there are no facts in his writings. Psychologists and psychiatrists are especially fond of the word. In 1980, the paradigm of a CP/M personal computer was a keyboard, Z80 with 64K RAM, 24 x 80 text display, and two floppy disks, each holding 256- to 512Kbytes. Naturally, all of the DOSes of the day featured sequential files for the simple reason that only a little bit of the data on those disks could be loaded into RAM at a time.

The New Paradigm:
The $2500 Mac Plus and the $1000 1040 ST each come with a megabyte as standard equipment. Each comes with space for just a single floppy drive. Isn't that just terrible?

Uh, well, actually, no. Times change, and old paradigms slink off to the paradigm graveyard to be replaced by new ones. Here is the new paradigm: If your floppy holds nearly a megabyte and you have more DRAM than the capacity of that floppy then you do not need a second floppy! Not for a personal computer, that is.

Why? Very simple. A utility can be written which can copy a floppy disk using a single step. Read the old disk into RAM. Write to the new disk. Done. Or write to another new disk. And another.

Hardware evolves with astonishing swiftness. All that hardware is totally useless without software. Software, especially operating systems, evolves with glacial slowness. So although both Apple and Atari are making 68000-based systems which are useful with a single floppy, all the operating systems out there are still based on pulling a little piece of data into local RAM at a time. The OS thinks the year is still 1980, when floppies held 8 times more data than the total available RAM. Have you ever watched a personal computer running an ISAM-type data base program? The floppy grinds and it grinds and it interminably grinds away. Hours later—many hours later—the program is finished.

Obviously, the way to run a personal-computer database program today is to load the entire floppy into RAM and then let your 68000, with its huge linear address space, massage that data damn quickly. Naturally, this technique is unavailable to the MS DOS folks for a number of reasons, but then the PC is 1981 hardware, not 1986 hardware. But the new paradigm is so very new that most folks have not caught on yet, even in the 1040 or Mac+ world.

I Ain't Guessing!
How do I know that a single floppy is usable over the long term? Simple. I have been using one constantly for over six months now! Sitting beside my host computer is a FLOPPY DISK SERVICES case-cum-power supply for two half-height 5.25 inch floppies. There are two HD floppies in that case. I have never used that second floppy, even though it would take only two hours, at the most, to modify the current HBASIC DOS to recognize that drive.

A New, Better DOS:
I have seen smart, experienced computer types scratching their heads over the way HBASIC handles data files. That's because they are still locked into the old paradigm where the disk is bigger than RAM. A lot bigger! Here's how one loads a 720K

LUCID, INC.
Learning UNIX/C In-Depth
"FOR THE CLARITY YOU EXPECT FROM TRAINING"
Ask about our HANDS-ON course for "C" Program Development Under MS/DOS"
For Free Brochure, Information or Registration Please Write or Call:
Alice Moss — Director
LUCID, INC.
260 Fifth Avenue
New York, N.Y. 10001
Suite 901
(212) 685-7106
Unix is a trademark of AT & T Bell Labs
MS-DOS is a trademark of Microsoft Corp.

Circle Number 59 on the Reader Service Card

Sort RM/COBOL Files
• • • QISORTC • • •
Finally, a sort utility for RM/COBOL.
Using efficient algorithms in assembly language making it the fastest sort on the MS-DOS market. Qisortc supports:
- RM/COBOL Version 1.5x & 2.xx files
  - Single / Dual File Indexed
  - Relative and Sequential
- ASCII, COMP-1, & COMP-3 fields
- 32 sort control fields
  - ascending / descending
- 32 record selection fields
  - EQ:NE:GT:GE:LT:LE field:constant
  - implied AND, OR may be specified
- 32 output fields or total record
- CALL from RM/COBOL, when provided subroutine is linked with runtime
- Use pathnames and return codes
- Keyboard or Parameter file
- Requires MS-DOS ver 2.xx or above
- Only $95.00 - COD, VISA, or M/C

Quantumn Information Systems, Inc.
145 N. W. 85th St., Suite 103
Seattle, Washington,98117
(206) 789-2888 Telex 4941041

Circle Number 31 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

NEW

FILEMOVER

by California Jack Cassidy

$59.95 *(IBM PC, XT, AT or clone, 256K min.)*

Not copy-protected, includes Source Code

FILEMOVER™ is perfect for Copying or Moving all types of files from disk to disk (or directory to directory). Reorganize your disk library and make back-ups without hassle. FileMover is easy to use and menu driven. Hard disk compatible too. **Subdirectory structure is preserved** when files are copied. And, if your destination disk fills up, you can continue on another disk.

FILE SORTER: Automatically alphabetize directories by file name and/or extension. Transfer files to other disks in any order.

FILE DISGUISER: Hide/unhide files so only you know they exist. Alter file names, dates and times. Convert files to Read-Only status. Inspect files in Hex-ASCII format...

QUICKSORT: DOS's sorter is extremely s-l-o-w on anything but small jobs. FileMover's sorter is one of the *fastest* anywhere!

Bonus–FOREVER FILE DELETER

Did you know that files you have "deleted" often *stay on your disks*, just waiting for someone to snoop through or undelete? FileMover's Super-Deleter totally ZEROES OUT unwanted files so they don't exist. Period.

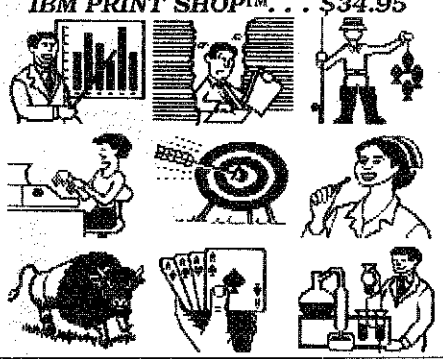
Free Turbo Command Chart

All of those *Turbo Pascal™* functions and procedures on one chart that always stays in front of you. A great PC programming tool.

NEW

MINIPIX DISK#2

200 New Graphics for IBM PRINT SHOP™. . . $34.95



Beagle Bros MICRO SOFTWARE

3990 Old Town Ave. / San Diego, CA 92110
619-296-6400

Products available at Software Stores or by mail:
Order Toll Free (Mon.-Fri., 9am-4pm Pacific Coast time)
1-800-345-1750 (Calif: 1-800-992-4022)
Add $2.50 shipping ($5.00 overseas), $3 COD, 6% if Calif.
All items in stock and shipped immediately.

Circle Number 84 on the Reader Service Card

Super Power Compatibility

The POWER SYSTEM™ from PECAN has set a new standard for comprehensiveness in program development environments that you won't find anywhere else.

PORTABILITY: Whether you're an ardent beginner or a seasoned pro, THE POWER SYSTEM lets you design software that can run on virtually any mini or micro without having to rewrite the program. The program you write on one computer will operate on another— even if the two hardware systems are incompatible. Your software never becomes obsolete — even if your hardware does.

INTEGRATABILITY: The versatility and flexibility of THE POWER SYSTEM enables different users to exchange program modules freely with one another, even though the modules may be written in different programming languages.

LANGUAGE INTERCHANGEABILITY: THE POWER SYSTEM lets you customize the modules of your application in the most appropriate of Pecan's powerful languages including UCSD PASCAL, BASIC, FORTRAN-77, ASSEMBLER, or MODULA-2. Once you've created your multi-language program, the modules are linked automatically. The application then runs as if you'd written it all in one language.

THE POWER SYSTEM FEATURES:

• NATIVE CODE • SEPARATE COMPILATION OF UNITS • EXTENDED PRECISION ARITHMETIC • DYNAMIC MEMORY MANAGEMENT • EVENT-DRIVEN MULTI-TASKING • CONFORMANT ARRAYS • PROCEDURAL PARAMETERS • 8087 AND BCD SUPPORT (PC's AND COMPATIBLES) • LARGE FILE EDITOR • LIBRARY UTILITY • ON-LINE HELP SYSTEM • FILE MANAGER

THE POWER SYSTEM Only $99.95

For PC-DOS, MS-DOS, AMIGA, ATARI ST, MACINTOSH, RAINBOW, TANDY, STRIDE, as well as most popular 8/16/32 bit systems. Also available in VAX, UNIX, PDP-11 OS's & others.

THE POWER SYSTEM includes the language of your choice: UCSD PASCAL, MODULA-2, BASIC, ASSEMBLER and FORTRAN-77. Choose an additional language for only $79.95. Buy three and we will send you a fourth one *FREE!*

To custom-tailor software, PECAN offers such programming tools as:

CROSS-ASSEMBLER/NCG PACKAGE	**$79.95**	**SOFTEACH TUTORIAL**	**$49.95**
KEYED SEQUENTIAL ACCESS METHOD	**$59.95**	**MENUMAKER**	**$49.95**
CODE OPTIMIZER	**$59.95**	**PROGRAM ANALYSIS TOOLKIT**	**$59.95**
SYMBOLIC DEBUGGER	**$59.95**	**SORT UTILITY**	**$59.95**

Write for complete list of development aids.

ATTENTION CORPORATIONS: Call and ask about our training seminars.
SCHOOLS: Special Educational Discounts

NOT COPY PROTECTED

Mail Your Check Or Money Order To:
Pecan Software Systems, Inc.
1410 - 39th Street
Brooklyn, New York 11218
(718) 851-3100
ITT TELEX NUMBER: 494 8910
COMPUSERVE CODE: 76703,520

PECAN™
The UCSD Pascal Company

Credit Card Orders Call Toll Free **1-800-63-PECAN**
(NYS) 1-800-45-PECAN

Please add $2.50 for shipping within the US. Foreign orders add $10.00 and make payment by bank draft payable in US dollars on US bank. New York State residents add appropriate sales tax.

UCSD Pascal is a trademark of The Regents of University of California.

Circle Number 71 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

data file containing 92,160 8-byte floating point numbers into an Atari 1040 ST using HBASIC:

```
10 DIM A[90,1024]:DATALOAD filename,A[]
```

That is it! Finished! Done! When you are done massaging those 92,160 double-precision numbers you simply

```
900 DATASAVE filename,A[]
```

Prices of 1-megabit DRAMs are now where they were for 256Ks in the spring of 1984. If this trend continues, the crossover from the 256K chips to the 1-megabit chips might occur in the early fall of 1987. In the second half of 1987 a high-end personal computer could have a 32-bit processor and a minimum of 4 megabytes of RAM, expandable to 16 megabytes if one uses 128 chips. We will have completely filled the linear addressing range of a 68000! What I am talking about is only a little over a year away from reality (plus maybe a little production lead time).

And three years after that, the 4-megabit chips will arrive, so we will have a minimum of 16 megabytes, expandable to 64 megabytes if we use 128 chips. Folks, what I am describing is a personal computer! A high-end personal computer but a personal computer nevertheless.

It Just Happened!

On Jan 1, 1986, there weren't any personal computers with 700K+ floppy disks which had more RAM than disk capacity. DTACK boards and 520 ST upgrades and Mackish upgrades don't count; I'm talking about what you can buy across the retail counter. Both the 1040 ST and the Mac+ began selling across the counter on Jan 1. So the new paradigm has arrived very recently; you might not even have noticed it yet.

But this change is permanent; from this time forward (into the foreseeable future) personal computers are going to have more RAM than disk capacity and so will need only one disk drive. What is less obvious is that an operating system which efficiently uses "big chunk" files, not sequential files, is going to be needed in the personal computer environment.

Business computers, even the small ones that *PC Magazine* writes about, will always (into the foreseeable future) have more disk capacity than RAM and so will always need a DOS with sequential files.

I think we are seeing the end of an era. Up until now the arrogant, fatuous, stupid minicomputer folks argued that what they had (UNIX) was just what a personal computer user needed. More recently, *PC Magazine* could assert that the small business computers it writes about, the ones with all the slots filled and 76 megabytes of hard disk and ten co-resident utilities and tomorrow afternoon at 3 o'clock with virtual memory was just what a personal computer owner needed. The folks at *PC Magazine* have been doing that because, like the minicomputer folk, they don't know any better. (Isn't it obvious that *PC* should be named *PBC*?)

It took a year for cheap 256K DRAM to translate into mass-produced 1-megabyte rock-shooting toys (1040 ST) and 1-megabyte yuppie adult toys (Mac+). So it will be five years before 16-megabyte rock-shooting toys appear. Climb into your time machine and go forward five years. Now explain to an owner of a 16-megabyte 68040-based personal computer how absolutely vital it is that his DOS have sequential files. . .

(Actually, sequential files are not, and never have been, wonderful. Like toilet tissue, sequential files are a regrettable necessity.)

Word Perfect?

This junk mailer resides on Word Perfect 4.1 files. All of the text up until this point was originally entered on the Eagle II at home and transferred to the IBM via RS232 at 1200 baud. Naturally, all of the formatting codes had to be re-entered, a miserably time-consuming task. So I'm going to have to get me an IBM PC for use at home. Sigh.

Word Perfect is widely regarded as among the very fastest of the IBM word processors. And the 8088 is much faster than the Z80 the Eagle uses. Yet Word Perfect is at least 2.5 times slower than Spellbinder running on my Eagle! Looks like I'm going to have to retract my assertion that the IBM PC had several word processors written in assembly. And it looks like I'm going to have to buy an AT, not a PC, because I can outrun WP's cursor in certain modes.

Jim Seymour isn't always wrong. In *PC WEEK* he asserts that Word Perfect and Microsoft Word are going to divide up the biggest part of the PC word processor market. I think he's right. But it really worries me that Word is widely regarded as being a lot slower than Word Perfect. . .

HBASIC Report:

I have finished implementing general expressions. There are no known bugs at this time, but I have just begun exercising the various possible combinations of expressions, nested arrays and functions, etc. This time I'm doing this by writing a program which calculates the value of successive expressions and compares that value with predetermined correct results. Sometimes when they don't match it turns out that HBASIC is right and the predeterminer is wrong!

That's not all. There has been a big change in the way the strings and string functions are handled. They are now almost indistinguishable from Bellevue BASIC! The reason I don't like dynamic strings is that they spray the variables all over the place, go away for a long time for garbage collection, often have fatal garbage collection errors (Applesoft, Waterloo APL), and prevent contiguous machine code from being loaded, manipulated, and run as string arrays.

I had in the past believed that getting rid of those problems meant that strings had to work differently. Uh uh! So HBASIC strings still don't have the problems listed above, and yet they are almost identical in operation to Bellevue BASIC!

Here's what I did. Each Bellevue BASIC string or array element has associated with it a byte containing the length of the element and a pointer to the element. In some versions, there is also a down-link pointer stored with the element to speed up the garbage collection process. (I am most familiar with that in its Commodore BASIC 4.0 guise.)

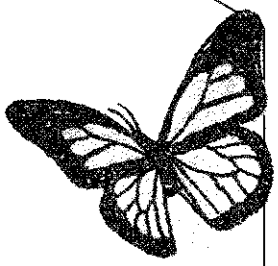
HBASIC had in the past a pointer and a maximum length byte associated with each string. In the case of string arrays, there was a single byte (actually a word) which specified the maximum length of all of the string elements. These have been retained. But a byte has been added which contains the current length of the string, just like Bellevue BASIC.

In the case of a string array, there is a "current length" byte for each element of the array. These bytes are not stored alongside each array element as in Bellevue BASIC because that would screw up using string arrays as executable machine code. So HBASIC uses a "shadow array" which is associated with

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Chalcedony PROLOG

Not Copy Protected



A REAL Clocksin & Mellish Prolog for BOTH major microcomputing operating systems — with full cross-compatibility. $99.95

Complete with the predicates necessary for POWER AI programming:

op () name () functor () clause () =..("Univ")

...And no constraining data typing.

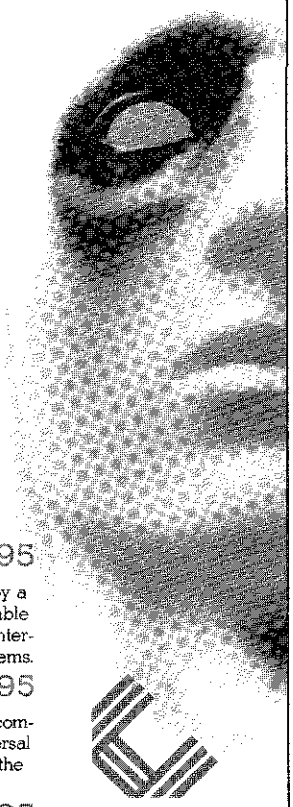
- Floating point
- Step-by-step tutorial
- Math functions
- Integrated editor

PROLOG/i

Extensible overlay library, 8087 support, large memory model (up to 640K)

PROLOG/m

Complete Macintosh environment with extensive pull-down menus and dialogue boxes.



No Risk Offer: *Examine the PROLOG/i or PROLOG/m documentation at our risk for 30 days. If not completely satisfied, return with disk still sealed for refund.*

APPLICATIONS — *Complete with SOURCE CODE*

NFL X-pert $49.95

A true interactive expert system written by a professional knowledge engineer. A valuable learning tool for any Prolog programmer interested in using Prolog to develop expert systems.

TOOLBOX $29.95

More than 50 subroutines that speed and compress list handling, searches, sorts, and reversal algorithms. An inside look at the tricks of the professional Prolog programmer.

TOYBOX $29.95

Written by an academician to help his students understand Prolog, this collection of puzzles and mind-teasers will illustrate how the Prolog programmer creates programs that find the best solution to the problem. Turn your computer into a super reasoning machine!

CHALCEDONY SOFTWARE, INC.

5580 LA JOLLA BLVD.
SUITE 126
LA JOLLA, CA
92037
(619) 483-8513

System Requirements:
Minimum 256K RAM (320K recommended) PC DOS/MS-DOS ANSI Standard Support
Minimum 512K Macintosh Macintosh-plus and HFS Compatible

SAVE 10% when you buy either PROLOG/i or PROLOG/m and all 3 applications.

PHONE ORDERS: 1-800-621-0852 EXT 468

☐ **PAYMENT ENCLOSED** $ _____
CA residents add 6% sales tax

☐ **CHARGE MY:** ☐ MasterCard ☐ Visa

Card No. _____ Exp. Date _____

Signature _____

Mr./Mrs./Ms. _____
(please print full name)

Address _____

City/State/Zip _____

PROLOG/i	**$99.95**
PROLOG/m	**99.95**
Check:	
MS-DOS ☐	Mac ☐
NFL X-pert	49.95
TOOLBOX	29.95
TOYBOX	29.95
Complete Pack	188.82

SHIPPING:
$ 5.00 U.S.
7.50 Canada
10.00 Carribean, Hawaii Air
20.00 Overseas Air

Circle Number 64 on the Reader Service Card

each string array. The shadow array has the same dimensions as the string array except that each element is a single byte, where the string array itself can have 1 to 255 bytes for each element. (No, that doesn't create undue overhead in locating string array elements—the address of the "shadow array" element is essentially "free" at the penultimate step of calculating the array element address.)

HBASIC retains the property that each string or string array element has a maximum length. That's why HBASIC strings are fixed in memory. But the entire objective of being compatible with Bellevue BASIC is to simplify conversion of programs written for the BASICA or Applesoft environment into HBASIC. So we simply change the default maximum string length to 255! Since Bellevue BASIC's strings can't be longer than that, we are now fully compatible!

"Aha!" you exclaim? "But that is going to take up a lot more memory space!" True. But what we are talking about is importing programs written for machines with less than 64K combined program and data space into an environment which will never be smaller than 512K and will usually be a megabyte or greater! So who cares?

Besides, that default length is (now) programmable and each string array's maximum element length is separately specified by the DIM statement, as it was in the past. Voila! Strings which are (nearly) indistinguishable from Bellevue's, except that ours work better!

It is somewhat hilarious that HBASIC, which started out being almost totally unlike Bellevue BASIC, is winding up a lot more compatible than, for instance, Software Limited's compiled BASIC.

Have no fear. HBASIC is not ever going to store and retrieve numeric data to and from disk by PRINTing and INPUTing each numeric data element . . . □

Threatened with another month of graveyard shift in D.P., Priscilla decides it's time for the personnel manager to learn the meaning of slice architecture.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Unix Books, Part 1

by Frank D. Greco

Why Unix? Whether we like it or not, the Unix operating system is becoming more and more prevalent. Many programmers have reached the outer limits of MS DOS's (documented) single-tasking capability and need the facilities of a powerful multi-tasking system. In addition, several large client bases mandate that RFPs (Requests For Proposals) require Unix or a Unix-like operating system. The United States Government and General Motors are two such large clients standardizing portions of their business around AT&T's latest version of Unix, Unix System V (System Five). Just these two giants alone represent potentially lucrative contracts to many software developers and computer consultants.

Unix's Biggest Advantage

Why are many government and corporate entities attracted to Unix? The biggest advantage of the Unix operating system is that within reasonable limits it's portable among computer vendors and machine architectures. Of course the mere fact of using Unix *does not* guarantee portability. Since a reasonable degree of portability is Unix's biggest advantage, let's delve into this concept of portability a bit.

Did FORTRAN Inspire Unix?

We can compare the portability of Unix with the portability of FORTRAN. FORTRAN was developed primarily so that programmers would have a common language in which to communicate with a computer; they would not have to learn a new Assembler language whenever a new machine came on the market. To help clarify matters for FORTRAN programmers and FORTRAN compiler writers, the ANSI standards group got together and formulated rules for a standard FORTRAN. So long as the programmer used *only* ANSI Standard FORTRAN conventions on a compiler classified as an ANSI Standard FORTRAN, he or she was guaranteed of portability. This made sense; however, someone forgot to specify a standard operating system interface (file system manipulation, device I/O, resource allocation, process execution, etc.). FORTRAN compiler vendors tried to supply this deficiency in their *own* diverse ways. Now we have at least ten major flavors of FORTRAN, all of them compatible (portable) *to a degree* with the others.

Unix Came from Space

Unix didn't have as altruistic a beginning as FORTRAN: Bell Labs inventor Ken Thompson needed a vehicle to "port" his GECOS-based Space Travel game. But subsequent versions of Unix evolved a similar standard operating environment for programmers. During a one-year sabbatical at the University of California at Berkeley, Thompson waved the Unix flag (it wasn't called Unix then). Several bright graduate students were so interested in the topic that they, along with some financial backing from the U.S. government's Defense Advance Research Projects Agency (DARPA), increased Unix's efficiency and functionality. This resulted in the BSD (Berkeley Software Distribution) version of Unix, which is currently BSD 4.3.

Let the Games Begin

About five years later, Microsoft acquired a $43,000 Unix source code license and added their own enhancements—most importantly, some interprocess communication primitives to help implement user-level file/record locking. Microsoft named their Unix "Xenix" because AT&T Licensing requires that resellers not call their products Unix (a mistake in my opinion). Other vendors followed. We now have Venturecom's Venix (non-standard real time enhancements), IBM's PC/IX (allows only small model 8086 programs), DEC's ULTRIX (based on BSD 4.2), AT&T's System V (adds new interprocess communication primitives), IBM's AIX for the RT (adds a new layer *below* the kernel), etc. We have at least ten major flavors of Unix, all of them compatible (portable) *to a degree* with the others. The Unix kernel and its utilites are rich enough that the *majority* of applications don't infringe on the non-portable areas at all. To its credit, the C programming language is mostly responsible for this. So while most Unix proponents push "portability" as a Unix asset, it's the portability among different manufacturers that is the big draw. You can take your C-written Unix applications and buy the machine with the best price/performance statistics. And that is why IBM dislikes Unix, or what it represents, and is probably *not* going to strongly back the operation of a *standard* (that's read "a pure AT&T strain") Unix. One look at the pricey and non-standard IBM PC RT port of Unix, the $3,400 AIX, and you'll see what I mean; but that's a topic for another article.

Now that I've finished my diatribe on Unix's raison d'etre—portability among machines—here are some of my unabashedly subjective opinions of the popular Unix books on the market today plus a sprinkling of some of my own Unix tips. A "report card" similar to those you see in software and hardware reviews appears at the end of each book review.

Lomuto and Lomuto
The Unix Primer
Prentice Hall

The Lomutos have put together one of the best Unix books for the computer neophyte. Aimed specifically at the pure beginner (especially with a clerical background), this primer covers using an editor, basic file and directory maintenance, text processing, and a few Unix utilities. About fifty percent of the book dwells on the Unix line editor "ed" and the standard Unix text processor "nroff." Nowadays, most Unix end users and programmers will rarely, if ever, use these archaic tools. End users usually go for a more usable screen editor like INed (a Rand editor derivative from IBM's partner-in-arms Interactive Systems) and a *real* word processor like CrystalWriter. Programmers new to Unix will take to either vi or EMACS or some newer generation editor. Even so, learning ed can still be of value to the Unix programmer. Since ed reads from standard input, you can redirect input to it; therefore, "canned" editing scripts, which occasionally come in handy, can be created. For example,

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
%  cat ed.script                # Display the "canned" script

# Run 'ed' on 1st command arg, redirect ed's responses to
# the bit bucket (/dev/null) and take the input from right
# after'EOF' until the next'EOF' (EOFisjust a convention).
# Locate the line that has the string "Date:" followed by
# any string (i.e., ".*") , change it to "Date: " followed by
# the current date and write it to disk and quit.

ed $1 >/dev/null <<EOF
/Date:.*/
c
Date: 'date'
.
wq
EOF

% chmod 755 ed.script          # Make it executable
% ed.script file1              # Execute the script
```

Another reason for learning the basics of ed is that the concept of a "regular expression," which ed uses quite a bit, comes up in other Unix utilities like "grep," "expr," and "sed." There is even a library routine for the Unix C programmer that accepts regular expressions and performs string matching (see regcmp(3) or regex(3) in the *Unix User's Manual*).

Although the Lomutos provide a good conceptual overview of the Unix shell, the meager coverage of the shell makes me wince a bit since knowing the shell well is normally tantamount to knowing Unix well. One of the biggest stumbling blocks in using Unix for the first time is the shell interface. Once during one of my classes when I was explaining the use of the Unix utility "cat," which is the effective equivalent of MS DOS's "type" command, a student inadvertently left off the filenames to be displayed and typed:

```
$  cat
```

The terminal didn't respond with an output or error message, and the student just waited for a reply. He thought the machine had crashed and was ready to power the machine off (a dangerous measure on most Unix systems). In fact, nothing was "wrong"; the cat program sensing there were no specified filenames was merely *waiting* for input from standard input (by default, the terminal keyboard) before displaying the output to the standard output (the terminal screen). The MS DOS type command, if given no filenames, will helpfully reply with the error message, "Invalid Number of Parameters." A simple shell program (a "script") could easily make Unix's cat utility look like the MS DOS type command. Note that the pound sign/hash symbol (#) introduces a comment to the Bourne shell:

```
%  cat type

# Bourne Shell script to simulate (and slightly enhance) the
# execution of the MS DOS 'type' utility.

if test $# -lt 1
then
     echo Invalid Number of Parameters
     exit 1
else
     for i
     do
          echo "================ Filename: $i ================

          'cat $i'

               "
     done
fi

%  chmod 755 type          # Make it executable
%  type file1 file2        # Run it with two arguments
```

While there are few illustrations in the *Unix Primer*, varying fonts are used to good effect. Authors out there who intend to print their next books in the draft quality mode of an old Epson printer should look at the professional quality and feel of the *Unix Primer* to get an idea of how the print should look.

Since this book does not assume prior knowledge of computers or computer systems, I wouldn't recommend it to *real* programmers. But for computer beginners starting out on a Unix based machine, I give this book high praise.

Meets Intended Audience	**10**
Title Matches Contents	**10**
Accurate	**10**
Thorough	**7**
Use of Illustrations	**8**

Strong Points	*Illustrations, Concepts well explained*
Weak Points	*Minimal shell coverage*
Comments	*Excellent for the computing beginner*

Waite, Martin, and Prata
Unix Primer Plus
Howard W. Sams

The foreword to this Waite Group book closes with an odd request: "Leave the book lying in a prominent place. Your friends and other visitors will be very impressed." This kind of thing just perpetuates the old myth that Unix is accessible only to Bell Labs technicians!

It's also odd that the authors demonstrate some utilities that reside only on Berkeley's BSD 4.2 or the more vintage Version 7. For example, they talk about the standard prompt "% ," the "finger," "uptime" and "more" utilities, the "~ " macro, and the .cshrc and .login files. These are of Berkeley origin. Now that's not bad in itself (I prefer the usability and power of BSD 4.2/4.3 myself), but Waite and company should at least tell the reader in the introduction and inform the potential buyer in the synopsis on the back cover. Lapses like this confuse newcomers to a Unix system, especially users with backgrounds in MS DOS. These portability issues should be thoroughly explained to the reader. After all, this is supposed to be a primer.

The authors continually refer to Unix utilities as Shell Commands. On page 45: ". . .the commands date, cal, who, and finger. These are examples of 'shell commands,' which are tandard commands recognized by the shell program . . . ." This is misleading. The commands mentioned are merely utilities that reside on disk. The shell does not "recognize" them any more than if you said "multiplan." What actually happens is this: The shell looks at the first group of non-blank characters and considers that to be your request. If the group is not one of the several commands that the shell itself recognizes internally (such as "cd," "exec," "set," or "export"), the shell then looks in specified directories (indicated via the PATH shell variable or the default /bin directory) to try to locate the request file and execute it.

In discussing redirection, the authors fail to mention the concepts of standard input, standard output, and standard error. While they talk about redirection to a file or to another command, the important notion of *different* input streams and output streams is missing. Standard Error is *not* mentioned at all. This again gives their discussion a BSD flavor as the C Shell cannot redirect standard error. To compound the issue, the

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

authors say that file redirection is a feature of Unix. On the contrary, redirection to/from a file is a property of the shell. While Unix provides the low level facilities to implement file redirection through manipulation of the file descriptor table, it is the shell that manifests the redirection concept to a user.

I like the section on the vi screen editor; it's well done. But it made only minimal mention of the shell language, on only one page.

Color and boldfacing complement the text nicely throughout the book, which features a liberal use of cartoons. I thought the cartoons were in the wrong spots, and their punch lines were way out of context. Comic relief is fine if it is used appropriately. However, while a student is learning the vi editor, he or she doesn't need the distraction of a cartoon (p. 153) that has one youngster saying to another, "I'll trade you my serial interface card for two Z80 cards and a Reggie Jackson." In general, I found the *Unix Primer Plus* too "cutesy" to be of practical use.

Meets Intended Audience	7
Title Matches Contents	8
Accurate	4
Thorough	3
Use of Illustrations	6

Strong Points	*Coverage of vi*
Weak Points	*No shell programming mentioned, pointless cartoons*
Comments	*Emphasizes Berkeley utilities, Many illustrations but not always pertinent*

Groff/Weinberg
Understanding Unix—A Conceptual Guide
Que

Groff and Weinberg have managed to cover all of Unix without sacrificing clarity or content—from "A Brief Unix History" to "The Future of Unix." The guide is extremely well written and geared mainly for the technical manager as opposed to the nuts-and-bolts programmer. It states its intentions early and covers material topically. A companion book, *Understanding Xenix: A Conceptual Guide*, does an excellent job of discussing the Microsoft product licensed from AT&T. While most of *Understanding Xenix* is similar to (if not a mirror image of) *Understanding Unix*, Groff and Weinberg do cover many Xenix-specific topics to make the additional purchase worthwhile.

Relatively technical issues such as disk storage allocation, disk buffering, process management with the system calls fork( ) and exec( ), and interprocess communication are explained clearly in surprisingly few paragraphs. This book does make the assumption that you are relatively computer knowledgeable. Terms such as "process image," "swap area," and "blocksize" are bandied about with the assumption that the reader knows what they mean. Although many details were left out of this book, and understandably so, I strongly recommend it for project or technical managers.

Meets Intended Audience	10
Title Matches Contents	10
Accurate	10
Thorough	9
Use of Illustrations	7

PC/VI

Full Screen Editor for MS-DOS (PC-DOS)

Looking for an Ultra-Powerful Full-Screen editor for your MS-DOS or PC-DOS system? Are you looking for an editor FULLY COMPATIBLE with the UNIX*VI editor. Are you looking for an editor which not only runs on IBM-PC's and compatibles, but ANY MS-DOS system? Are you looking for an editor which provides power and flexibility for both programming and text editing? If you are, then look no further because **PC/VI** IS HERE!

The following is only a hint of the power behind **PC/VI**: English-like syntax in command mode, mnemonic control sequences in visual mode; full undo capability; deletions, changes and cursor positioning on character, word, line, sentence, paragraph or global basis; editing of files larger than available memory; powerful pattern matching capability for searches and substitutions; location marking; joining multiple lines; auto-indentation; word abbreviations and MUCH, MUCH MORE!

The **PC/VI** editor is available for IBM-PC's and generic MS-DOS based systems for only $149. For more information call or write:

Custom Software Systems
P.O. Box 678
Natick, MA 01760
617-653-2555

The UNIX community has been using the VI editor for years. Now you can run an implementation of the same editor under MS-DOS. Don't miss out on the power of **PC/VI!**

*UNIX is a trademark of AT&T Bell Laboratories.

Circle Number 66 on the Reader Service Card

TURBO PASCAL GENERATOR

THE GTP PROFESSIONAL MODEL

Generate error-free Turbo Pascal source code for:

DATA BASES MENUS
SCREENS REPORTS

Build complete, working programs in minutes!

FEATURES
- Indexed Data Bases
- Multiple Screens
- Automatic Updating
- Built-in Edits
- Retrieval Facility
- Automatic File Build
- Quick Screen Handling
- Speedy DB Access
- Context Sensitive HELP
- Full Keyboard Support

EASY
1. Paint-the-screen
2. Define fields & calculations
3. Generate, compile & RUN

FLEXIBLE
- 100% Turbo Pascal
- Modify in Pascal under Turbo's Editor
- Extensions are easy with 130 page Programmer Reference manual
- 100 documented source code routines included

NOT Copy Protected NO Royalty Fees NO Run-time Library Required

Requires: IBM PC (100% compatible) 256K RAM 2 Disk Drives
PC-DOS 2.0+ Turbo Pascal 3.0 from Borland International

PRICE $200.00
CALL FOR QUANTITY DISCOUNTS
VISA/MC – Check – Money Order – No COD or Purchase Orders
Texas Residents: add $12.25 Sales Tax
Outside US & Canada: add $25.00 Air Postage and make payment by credit card or money order in US Funds.

A E F
ALLEN, EMERSON & FRANKLIN
P.O. Box 928
Katy, TX 77492
(713) 391-8570

Circle Number 53 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Strong Points *Process Management explained well*
Weak Points *Minimal Berkeley information*
Comments *Excellent book for Unix concepts for the technical manager*

Kernighan/Pike
The Unix Programming Environment
Prentice Hall

Brian Kernighan (pronounced "Kerni-han") and Rob Pike are members of the creme de la creme AT&T Bell Laboratories at Murray Hill, New Jersey. The Murray Hill Labs hosts an amazing seven Nobel Prize winners.

Both authors (along with colleagues Ken Thompson and Dennis Ritchie) work in the Computing Science Research Department, where extremely interesting and truly state-of-the-art research takes place. Brian Kernighan is the co-author with C inventor Dennis Ritchie of the classic C language text. Mr. Kernighan also co-authored with P.J. Plauger the classic computer science texts *Software Tools* and *The Elements of Programming Style* (the title is a takeoff on Strunk and White's classic *Elements of Style* for writers). The occasionally eccentric and part-time physicist/cosmologist Rob Pike is the creator of the elegant windowing environment/communications software that runs on the patented Teletype 5620 DMD bit-mapped terminal a.k.a. the BLIT. This terminal was built by former Bell Labs colleague Bart Locanthi, who currently works for Silicon Graphics. If you're interested in the software that runs on the patented DMD 5620, it was discussed *in detail* in the ACM SIGGRAPH 1983 Conference Proceedings article "Graphics in Overlapping Bitmap Layers."

In the preface, the authors say, ". . .the goal of the book is to communicate the Unix programming philosophy. . . .based on the relationships between programs, . . ." Kernighan and Pike (K&P) never deviate from this goal and constantly remind the reader of the importance of program reusability and the concept of software tools. In order to use Unix well, the programmer must recognize these notions as vital to the entire development process.

As a trivial but illustrative example, consider the Unix utility "echo." Echo is either an executable utility program that resides on disk or, more recently, a built-in command of the shell interpreter. Its purpose in life is to echo, or print, its arguments. For example,

```
%  echo 3B or not 3B, that is the question
```

results in

```
3B or not 3B, that is the question
% _
```

At first glance you might ask, "What's echo for? I can surely see what I want to echo right on the command line!" My stock answer is that echo has no purpose—by itself. Of course you can see what would be echoed; you typed those words! However, when you combine a little Unix utility like echo with others such as "ls," "cat," "wc," etc., along with shell programming constructs such as "while," "for," "read," etc., in a file (called a shell script), echo becomes the print statement for the shell. This is very similar to the echo internal command of the MS DOS shell command.com. I wonder where they got the idea?

To see another example, look at the following small but exemplary shell script:

```
%  cat getime        # Display the contents of file getime

# Get the current clock time only

set 'date'                  # Assign the output of the Unix date
                            # utility, each token to a positional
                            # parameter (i.e., $1, $2,...)

echo The time is now $4     # The time is the 4th argument of the
                            # the output of the date command
```

Running this little shell script results in

```
%  chmod 755 getime  # First make it executable (done once)

%  getime
The time is now 9:29:11
```

Notice how the little echo utility becomes the print statement for the shell script "getime."

K&P go into more substantial and detailed examples of program interactivity that a programmer needs to know about in order to use Unix well. In the solid 100 pages describing the shell and shell scripts, they keep reminding the reader of the notion of reusability. In doing so, they manage to explain the useful filters "grep," "egrep," "awk," "tr," and "sed," along with shell programming constructs, rather thoroughly.

The most useful and interesting part of this book is Chapter 8, "Program Development." In this chapter K&P discuss the use and importance of the tools "yacc," "lex," and "make." They implement a relatively simple version of a C-like interactive calculator whose development utilizes these tools. The techniques learned in this chapter can be applied to formal language-code generation in interpreters, compilers, or other translators. (This is described at length in Prentice Hall's *Introduction to Compiler Construction with Unix*.) The one weak point, obvious by its omission, is the lack of an in-depth discussion of either of the standard Unix debuggers "adb" (A DeBugger) and "sdb" (Symbolic DeBugger). While these debuggers are certainly archaic (similar to MS DOS's DEBUG utility, but harder to use), they come with Unix or at least are bundled with a developer's package. K&P allocate a mere half page to demonstrating the use of these two dated debuggers.

Every now and then, you glimpse the authors' dry humor—true Bell Labs style. Consider this quote from the "Program Development" chapter: "This section describes the implementation of hoc1, a program that provides about the same capabilities as a minimal pocket calculator, and is substantially less portable."

This book is definitely geared to the advanced Unix systems programmer. Despite the first chapter's title, "Unix for Beginners," it is not. In reading the chapter, I get the feeling that K&P did not enjoy writing this obligatory introduction. Right after it, they accelerate into the meat of the matter, leaving the true beginner in the dust. Chapter 2 starts off by using the Octal Dump (od) utility to display the bytes of a file. This speed is comfortable for a system programmer, but not for the beginning application programmer. Be forewarned.

I found the "History and Bibliographic Notes" subsection at the end of each chapter interesting. These are historical references, anecdotes, or personal comments about the Unix operating system. Discussing the "awk" pattern scanning/pattern processing language, Pike quips, "awk was designed and implemented by Al Aho, Peter Weinberger and Brian Kernighan . . . . Naming a language after its authors also shows a certain poverty of imagination."

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

The Unix Programming Environment is well written but unfortunately lacking in useful illustrations. However, you have to have it if you are doing serious work with C or Shell programming development using the Unix operating system.

Meets Intended Audience	**10**
Title Matches Contents	**10**
Accurate	**10**
Thorough	**7**
Use of Illustrations	**1**

Strong Points *Description of tools (lex, yacc, make, C) for developing an interpretive calculator*

Weak Points *Miniscule amount of Berkeley Unix information, Sparse to non-existent illustrations, Extremely dry reading in places, Lack of a decent discussion of adb or sdb*

Comments *Very good book for Unix-based system developers, On my "must have" list*

Thomas/Yates
A User Guide to the Unix System
McGraw-Hill

This was one of the first books published to explain the Unix operating system to the universe outside Bell Laboratories. There are two editions of this book: one dated 1982 and the second 1984. I've heard that the authors have grossed over $500,000 with this book alone because of the age and timing of the first edition. However, despite its popularity, there are a lot of things that I don't like about *A User Guide to the Unix System*.

Thomas and Yates make some unusual statements in the chapter titled "A History of the Unix System": "In 1979, Bell Laboratories introduced the Unix system Seventh Edition and dramatically decreased the single-user binary license fee. This permitted computer manufacturers to economically develop Unix software for use with licensed Unix systems, and small businesses to license the Unix system for their personal computers." Personal computers? In 1979? I believe personal computers of that era were mostly based on either the 6502, Z80 or 8080 processors (Apple, PET, TRS, Exidy). While it was conceivably possible to run Unix on one of these older chips, it wasn't practical back then. Memory limitations, both primary and secondary, were severe.

Thomas and Yates also make a common "complaint" about Unix (usually heard in MIS-type publications like *ComputerWorld* or *Datamation*): "The Unix system lacks some essential features for business applications. For example, the command names are obscure; grep, pwd, and cat do not convey a command function." Now wait a minute—let's not get carried away. First of all, I'm not sure what Thomas and Yates mean by "lack[ing] some essential features for business applications." Unix lacks certain "business" features no more than MS DOS does, or MVS does, or VMS does, or Pick does, or AOS does, or VM/CMS does, etc. Pinning that rap on Unix is not fair. Let's be honest—Unix does have some deficiencies, but so do other operating systems! Secondly, the utilities grep, pwd and cat don't convey command functions only if you are not familiar with them. To someone experienced with Unix, grep has a perfectly valid name (Globally look for a Regular Expression and Print it—GREP), pwd has a perfectly valid name (Print the Working Directory), and cat has a perfectly valid name (conCATenate and print).

FULL POWER DEBUGGING

DSD86, The PC-DOS Debugger *69.95*
DSD87, The PC-DOS Debugger with 8087 Support . *99.95*
DSD80, The CP/M Debugger *125.00*

SoftAdvances
P.O. Box 49473 • Austin, Texas 78765 • (512) 478-4763
1-800-232-8088

Circle Number 33 on the Reader Service Card

TURBO SOURCE SEARCH (TSS)

TSS is a **RBBS-by-mail** for **TURBO** Pascal. Stop paying AT&T, MCI, SPRINT et al. 10 times what it will cost you if you get it by mail from TSS.

Join **TSS** for $25. Then buy our library disks, **$2 per disk.** (45 disks and growing!!) Archived files = **700+ Kb** of code per **TSS** disk. **TSS** gives more bytes/buck! **Why pay more to get less?**

For your $25 you get
- 2 **FREE** library disks (utilities & catalog)
- 2 **FREE TSS** disks with your first order
- a validated password for the **online TSS-BBS**

TSS-BBS will be an "e-mail" drop-off for members to place orders then log off. It will also serve more traditional RBBS needs. Members will be able to DL the **TSS** disks **(at 10 times the cost of mails!)** if they really MUST have it now!

Join the **TSS BBS-thru-the-mail** for the best in public domain and shareware **TURBO** code.

TURBO SOURCE SEARCH
P.O. Box 876, Scituate, MA 02066, (617) 545-6677
payment: VISA/MC/COD **if validated**, otherwise check/money orders

Circle Number 35 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Since many people besides the authors consider Unix an unfriendly environment, let's look at this for a bit. A common complaint about the Unix operating system is that it is not "user friendly"; it is only programmer friendly. Most critics cite the non-mnemonic command names and lack of decent error handling. Sufficient mud slinging at the user interfaces of Unix and other operating systems (DRI's CP/M and Microsoft's MS DOS, for instance) warrants a repetition of two old maxims I teach in my operating systems course:

1. For most users, operating systems should be heard and not seen.
2. Operating systems are not supposed to be user friendly; they are supposed to be useful.

As the first axiom suggests, operating systems and related subsystems (command interpreters, programming languages, file utilities) should be directly used only by programmers, not by end users. These are power tools for the programmer. End users should use application software packages to do their work. Most drivers are oblivious to the internal operation of a car. Although a healthy understanding of its internal mechanisms will lead to more efficient use of the car, getting from Point A to Point B is the prevailing consideration.

Maxim 2 answers the so-called Unix experts who malign the Unix utility program grep (recall: Globally look for a Regular Expression and Print) as cryptic and difficult to use. An old CP/M command PIP (Peripheral Interchange Program) is (or perhaps was) often criticized for the same reasons. But these programs are intended for *programmers*, not general users. It is therefore understandable that most users have difficulties with these commands, but this does not justify the onslaught of criticism heaped on these *programmer* utilities. Unix and most other effective operating systems give programmers (not end users) useful software tools and an environment conducive to their work. If programmers have an operating system in which to develop, debug, and maintain their programs, better user application packages will be created. If a knowledgeable user does not care for the name of any Unix utility, he can simply create a shell script, shell alias, or even a C program to hide the Unix flavor of the command's name.

Many companies send their word processing, clerical, or administrative and management personnel to Unix workshop courses. For the most part, these people really should not be learning bare-bones Unix. They should not be learning to sh, nroff, sed, or awk. These workers have their own areas of expertise, which they should concentrate on. The spreadsheet user worries about What-Ifs concerning costs, inventory and profits; the programming user worries about What-Ifs concerning byte-ordering, interrupt vectors, and concurrent programming. So instead of learning about or using specific operating system mechanisms, these end users should capitalize on available application level tools written by competent Unix program developers—menu or icon driven filing systems (sometimes called visual shells), true word processor programs (not nroff, Unix's archaic though potent text formatter), and spreadsheet programs. People should not become computer literate; computers should become people literate. Unix is just another step towards this ultimate goal.

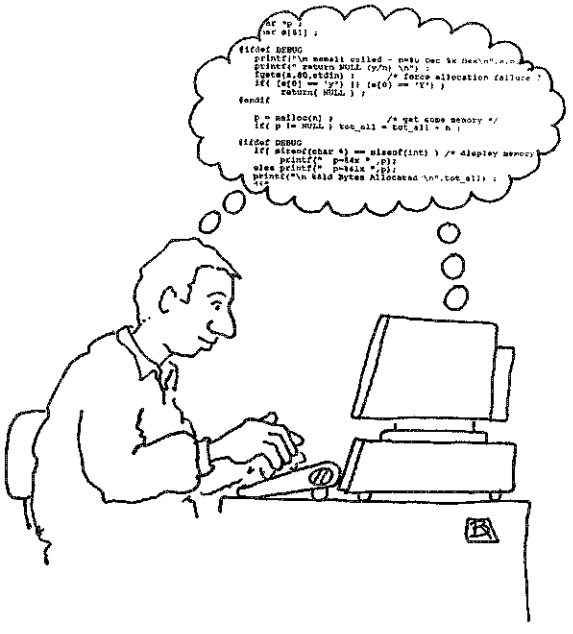
Thomas and Yates seem to enjoy using the "by example" teaching paradigm. They describe each utility in one or two sentences, then give many examples. This is not a bad methodology (in fact it is an excellent technique); however, it should be reinforced by subsequent *in-depth* explanations of the utilities. The authors fail to do this.

The all important shell interpreter is not explained well, shell programming is not even mentioned, the text formatters nroff and troff are mentioned only in passing, and regular expressions (used by ed, sed, grep, expr, and others) are neglected completely. I really cannot recommend this book.

Meets Intended Audience	5
Title Matches Contents	5
Accurate	3
Thorough	1
Use of Illustrations	1

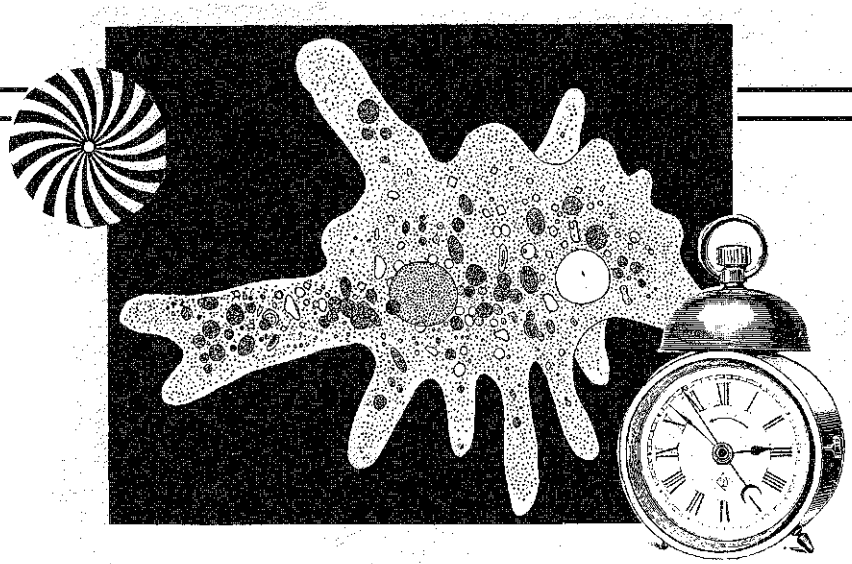
Strong Points	*Many examples to support explanations*
Weak Points	*Important topics either ignored or incompletely explained*
Comments	*Not for programmers, barely sufficient for end users* □

Frank D. Greco will continue his survey of Unix books in subsequent issues of Programmer's Journal.



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

How Many Amoebas at One O'clock?



by M.J. Budlong

Martin Gardener's puzzle column in *Scientific American* sparked many of us to mull over burning issues such as "Who owns the Zebra?" and "If Mr. Brown is sitting at the North side of the table, then exactly how old is his wife?" We did these puzzles as a form of escape. We all knew that if one morning we asked the postman how old he was he would never answer, "I have two sons. The sum of their ages is the square root of what my age will be two birthdays from now. The difference in their ages multiplied by 2 is equal to one-third of my age right now." None of us ever approached a waitress with the question, "If you were the other waitress, which door would you say is the door to the Men's Room?"

The world of the puzzle addict was populated by noblemen who always told the truth, and peasants who always lied, kings who chose advisers by painting red dots on their foreheads, and merchants who always carried a sum of cash that could be determined by resolving a complex polynomial. A guest at a puzzle world cocktail party identified himself, not by name, but by indicating that

A. He did not live next door to the man in the red house.
B. He frequently played bridge with the German.
C. His pet which was not a canary smoked only Havana cigars.

With the computer revolution, a lot of us who would have ended up as *Scientific-American*-reading-beach-combers could now get honest jobs. The exclusive OR gate entered on an arena in which this otherwise useless talent could be used to earn cold hard cash.

Suddenly someone was willing to pay money for a program that could deliver the answer to how many coconuts were in the pile before the fifth monkey took half the coconuts plus one half a coconut, 10 thousand times in one second—not for that precise problem of course, but from a logic point of view, one that amounted to the same thing.

After all, keeping an electronic recording of which monkey got the coconuts is desirable according to every IRS inspector I have met.

But now we have a logic machine that can prove millions of times per second that a man can outdistance a bullet simply by running one-tenth of the distance travelled by the bullet since the last interval was tested. Impressive? Yes, but just because I can force myself to think like an idiotic machine in spite of all real world evidence that people at parties do not introduce themselves by telling me the log base 10 of their shoe size, does it follow that a machine thinking idiotically, millions of times faster, can think like I do? The machine would crunch away and come up with a guest's shoe size in a fraction of a second, record it to magnetic media, and beam it via satellite data link to the International Shoe Size Database. I, on the other hand, might smile feebly and take off in the direction of the prettiest red-head in sight.

The long range goal of Artificial Intelligence might be to run the planet economically and socially from a multi-billion dollar computer that would presumably know everyone's shoe size and where that size of shoe could be bought. Such a computer would distribute shoes based on shoe size averages for planetary geographic areas, and the mystery cocktail guest, no longer concerned about squeezing size ten feet into size nine shoes, would reveal his real name.

If the AI buffs are trying to groom a megabucks machine to replace politicians in decision making, let me put their minds at ease. We don't need this expensive approach. A private study done on governments and world leaders revealed that based on the amount of useful work they do, and the amount of human intelligence and common sense they put into their jobs, the average politician can be fully replaced by a zip lock sandwich bag. We could retain the two-party system by using zip locks for one party, and the members of the other party could be replaced with rolls of Scotch tape.

If you ever want to play around with Artificial Intelligence, try this problem. Paint a red dot on your terminal just above the CRT (on the terminal's forehead) and write a program that will allow the computer to deduce the color of the dot using only the following information as input:

A. A hen and a half can lay an egg and a half in a day and a half.
B. 3 cannibals and 3 missionaries want to cross a stream in a two man canoe.
C. Colonel Mustard is in the library with the candlestick.

Even if you can't get the computer to solve the problem, leave the red dot in place as a permanent reminder of the true heritage of AI. □

M.J. Budlong is a programmer/systems analyst who writes a regular technical column for VS News. *As Chief Researcher for the Citizen's Commission on Human Rights, M.J. Budlong has written and contributed to articles in the* London Sunday Observer, Berliner Zeitung, *and Johannesburg's* Peace and Freedom *on drug abuse, conditions in South Africa's black mental hospitals, and the roots of twentieth century psychiatry.*

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Parents, Children, Redirection, and Piping with DOS Functions 45H and 46H

by George Defenbaugh

Overview

When the DOS functions 45H (DUP) and 46H (FORCDUP/CDUP) are discussed, words like "obscure," "unclear" and "mysterious" crop up—even in such well known publications as this magazine, *Dr. Dobb's Journal*, and books from Microsoft Press. Why are these functions so perplexing? Why do we hear only about the buffer flushing and directory updating ability of function 45H?

Because I play PC only part-time (I live on mainframes during the day), and because I write Assembler so slowly, I never seemed to have the time to research these questions. But once I moved to C (the language for frustrated Assembler programmers), the discovery of the freopen( ), fdopen( ), dup( ), and dup2( ) functions in Microsoft 3.00 C sparked my interest in the subject again. I decided that I needed to do more study. Hence this article, where we will look at some code fragments and discuss questions such as:

Why haven't *you* used DOS functions 45H and 46H yet?

Where did this stuff come from, anyway?

What do standard file handles save you?

Where does redirection fit in?

How do you use this stuff, anyway?

Why Haven't You Used Dos Functions 45H and 46H Yet?

MS DOS comes out of the box with 5 standard files already defined, which use the file handle numbers 0–4. Also provided for your programming enjoyment are DOS functions 45H & 46H. These are the DOS functions that duplicate or force the duplication of an existing file handle. They duplicate handles *only* now. They *don't* duplicate an existing file or allow you to do two separate input/outputs with one language statement. How many of you have ever seen a "real program" (i.e., not a code fragment) that used these functions? Don't be shy now—get those hands up. Few to none? You say that what you've seen are merely illustrations that show how to redirect a standard file handle somewhere else, like redirecting standard output to a disk file?

Would you believe me if I said that I use these functions in every one of my programs? (You wouldn't?) After all, why in the world would you go to the trouble of redirecting a standard file *inside* a program? If you want the file, why not just go directly to it? And since you have the standard file handles already identified, you don't even need two sets of code in case you want to decide at the last minute what you should do! You merely put the desired file handle in a variable accessible to all code that does any output.

Ah, there's the rub. This illustrates what I call the "single application program" (SAP) syndrome, a malady suffered by many who have grown up on PCs. (Many mainframe programmers suffer from this also.) The syndrome manifests itself by limiting the perspective of the programmer to just his or her single program or application. Usually a person writes a program and considers it a "child" process of the supervisor. Occasionally we get sophisticated—writing two programs and allowing them to somehow communicate to each other. But seldom do we ever consider things from the supervisor's, or parent's, point of view, much less write code that functions like a supervisor or parent. And the fact is, Folks, that redirection is for the parent, while standard I/O (not Trix) is for the kids. Although redirection and standard I/O have some value separately, their real utility surfaces when the concepts are combined. We'll see the combination in "How Do You Use This Stuff, Anyway?" below.

Where Did This Stuff Come From Anyway?

We all know that programs that need to access files have at least four different ways to determine which files to use. The filename(s) can be:

1. put inside the program when it is written (hardcode them),
2. put inside a configuration file (or equivalent) that can be read,
3. passed on the command line when the program is started,
4. attached to a pre-defined file handle (one that someone else established for us).

1 is the original "dumb" way, reminiscent of programming 20 years ago. It is not flexible, but is the easiest to do and does eliminate some potential runtime errors. 2 is more flexible and is seen by example in environment variables, .PIF files, etc. This is a good method if the filenames don't change too often. 3 is like 2 but a little more dynamic because it is easier to change the filenames each execution since they are supplied on the command line. 4 is best if we are dealing with only a few files and if we want to retain maximum flexibility. What, then, might we call these pre-defined guys? How about standard file handles?! The existence of pre-defined handles allows a program to always refer to its file the same way, regardless of how the file might be connected to the external environment.

What Do Standard File Handles Save You?

Let's assume you accept my premise that these standard handles have value. Kernighan and Plauger's book *Software Tools* is often mentioned as a source for the the concept of standard file I/O and redirection. *Software Tools* discusses the value of writing small programs (tools) that can become standardized while still encouraging variety and experimentation. Kernighan and Plauger also point out that the combination of these programs in novel ways can occur because we did not try to identify all the "obvious" connections up front.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

One of the valuable things about creating these programs is that we need not include in the program the code to "set up" the external file environment. We can let the supervisor or parent do that, and the program can get on with the job for which it is written. Certainly the program could have set up its own external files, though. Whichever does it, the things involved are to:

1. parse the command line or file for all filenames and symbols,
2. associate correctly the symbols (possibly < and >) with the filenames so that you can tell input filenames from output filenames,
3. open the file(s) in the correct mode, possibly creating them,
4. provide all necessary error messages and error handling,
5. strip the symbols and filenames out of the command line so that the rest of the program will not be confused by them,
6. close the file(s) when done.

This is pretty boring stuff after you do it once, just detracts from the job at hand, and also must be present in *every* program that can be executed by the parent. Why don't we just talk the operating system supervisor into taking these responsibilities away and letting us attend to more interesting work; i.e., let's get the parent to set up the file environment for us and connect the external files to the pre-defined (standard) file handles. Then all we (the program) have to do is access the file through the pre-defined handle—no fuss, no muss, no bother. (Remember that in DOS one program can act as a parent for another through the DOS function 4BH, the EXEC function call. This is how COMMAND.COM acts toward us.)

Where Does Redirection Come In?

By defining standard input and output we have eliminated much repetitive work associated with common program I/O. We also mentioned above that we planned to let the parent process hook up each file handle to the desired external file. This is because just establishing standard file handles means little by itself. Let's assume for the sake of discussion that the default file or device for standard input is the keyboard and that standard output is connected to the screen. What good is there in merely setting up an environment where the keyboard and screen, or any other two files or devices for that matter, are already defined? Might not many situations require that other files or devices be utilized? Well, if we can get that same parent who already established these files for us to "attach" the designated standard handles to files or devices of our choosing, we have created a very flexible tool indeed. This is exactly what redirection is all about. And having done that, we also might automatically transfer the output of one program to the input of another. Enter the DOS pipe, invoked by the symbol "|." How neat! All those tools we created, which dynamically have their input and output files determined at runtime, now automatically work with this new piping concept. All you do is upgrade the supervisor to handle the piping symbol. So redirection is really a feature implemented by an operating system supervisor or parent process and is used to facilitate modular system building. By using it along with standard file handles we have:

1. eliminated redundant code from many programs,
2. provided a facility for programs to communicate more easily with one another,
3. developed a simple, transparent (to the program) way to let the user determine what the input or output to the program should be each time it is run.

How Do You Use This Stuff, Anyway?

You can write a program, even in Assembler; utilize all the above concepts of standard file handles, redirection, and piping; and never even think about DOS fulctions 45H and 46H. This is because when you use the symbols "<," ">," or "|" at the DOS command line you cause DOS itself to use these concepts before your program is invoked. In the same way, if you are going to initiate a child process, you can alter the file environment of that child process by using these concepts before you do the invocation. That accomplishes the same thing for the child process in relation to you that you cause to happen to programs like FIND and MORE when you execute them from the DOS command line and use redirection or piping—for example, when you do

```
DIR| FIND "-86" > THISYEAR.DIR&&
```

to locate only files modified this year and to save them in a disk file called THISYEAR.DIR.

Let's look at the code fragments necessary to redirect a file handle and do other related things. We'll develop the code in Microsoft 3.00 C. Although we could have developed the code in Assembler, I think more people might be able to read C. The format will be question, answer, and explanation. We also will be mixing high-level and low-level C functions. This is normally not a "good" idea, but using high- and low-level functions together may have merit if you completely understand the effects. There's no law against it, but admittedly it isn't recommended. Consider these examples for illustration only.

Continued...

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

How do you redirect standard input (stdin) to disk, then back to the originally defined handle that standard input first came with?

```
int fh;                                 /* define an integer fh             */
fh = dup ( 0 );                         /* dup stdin & store cloned handle  */
freopen ( "in-file", "r", stdin );      /* open in-file to read; close stdin */
... code body one ...                   /*  & point stdin handle to in-file  */
dup2 ( fh, 0 );                         /* point stdin handle back at stdin  */
... code body two ...
```

The dup( ) function "clones" the pre-defined file handle 0 (0 is stdin) by giving us another handle that is a copy of 0 and saving the copy in the variable fh. This is because freopen( ) will close stdin before it points the handle 0 at in-file. And if we let freopen( ) close stdin, how would we ever open it again? What filename would we use? We *must* save it in order to restore things to the state in which we found them. "Code body one" is where we might initiate a child process by using one of the exec( ) or spawn( ) classes of functions. Note that the use of getchar( ) inside "code body one" will read the file "in-file" (i.e., stdin has now been redirected to in-file). Once the child process is done, we can restore handle 0 to the state in which we found it with the dup2( ) function. Rather than cloning the given handle as dup( ) does, dup2( ) "forces" the handle 0 to be the same as the one saved in fh. That one of course came from a cloning of 0 in the first place. Note that dup( ) and dup2( ) correspond directly to DOS functions 45H and 46H and that freopen( ) combines fopen( ) and dup2( ) into one function. Using getchar( ) in "code body two" will cause the keyboard to be read, just as it does normally.

A side note is that freopen( ) exists because not all implementations of C allow syntax like "stdin = fopen (...)." This is because stdin is not required to be an "lvalue," which is eligible to be used on the left of the assignment. The ANSI C draft standard explains this decision.

How do you redirect any high-level FILE access from file "in1" to file "in2" and then back?

```
FILE *fp1;                              /* define pointer to FILE structure  */
int fh;                                 /* define integer                    */
fp1 = fopen ( "in1", "r" );             /* fp1 now lets us read in1          */
... code body one ...
fh = dup ( fp1 - stdin );               /* clone the handle from fp1         */
freopen ( "in2", "r", fp1 );            /* close in1, open in2 but use the   */
... code body two ...                   /*  handle used by in1               */
fp1 = fdopen ( fh, "r" );               /* allow fp1 now to point back at    */
... code body three ...                 /*  in1 where it left off            */
```

Any access through the FILE pointer fp1 in "code body one" accesses the file "in1." We then clone the file handle associated with fp1 and store the new file handle in fh. This is necessary because the freopen( ) will close the file associated with fp1 and we would "lose" access to "in1." Access in "code body two" using fp1 references "in2." Finally, accessing "in1" again requires that the cloned file handle be used to create a FILE pointer, which can be used in high-level file accesses into "in1" again. It is important to realize that although freopen( ) closed "in1" as viewed through fp1, it did *not* change the file pointer as associated with the low-level file handle. When that low-level file handle is promoted into a stream pointer through the

object $99 * source $249 *

BUSINESS BOARD SYSTEM

The Business Board System (BBS) provides a dial-up information center that can be accessed 24 hours a day via modem at 300/1200/2400 baud. The multi-level menu system is easily customized with no programming required. An unlimited number of data entry forms are user configurable and can be presented to callers based on their menu selections. The business applications are unlimited! This package is written in compiled BASIC and includes both CONSOLX for remote PC operations and BORDM our data base file manager.

object $29 * source $99 *

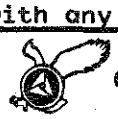
CASINO CARD GAMES

Treat yourself to these exciting video graphic card games just like you've seen in Vegas and Atlantic City! You get BlackJack, Slot Poker, and a "bluffing" game of Draw Poker! The package can run on either monochrome or color IBM PC/XT/AT & compatibles. It uses realistic card graphics and player "windows". BlackJack allows play variations and also displays statistics. Written in compiled BASIC. Our Windowing pkg included with source.

object $39 * c source $249 *

C DATA ENCRYPTION

A complete security system using DES (Data Encryption Standard) per U.S. government standard FIPS PUB46 written in Microsoft C. The 3 DES modes are included: Electronic code book, cipher feedback, and cipher block chain. The cipher key may be input in hex or ASCII (for operator convienience). Also included is data compression and telecommunications formatting functions which allow for faster transmission and storage with any computer or service. A menu driven batch file provides simple operations.



HAWKEYE GRAFIX Inc
3415 HYDE PARK DRIVE
CLEARWATER, FL 33519
813-786-8161
Data comm & terminal emulation
Windows & pop-up time/billing
* Prices discounted thru DEC 31

Circle Number 57 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

fdopen( ) function, access back into "in1" in "code body three" will *pick up where it left off* as far as the next characters to be read from the file are concerned. We can of course rewind the file so that reading can start at the beginning by doing a rewind (fp1).

Can you close a file if a duplicate file handle exists?

Sure. Each file handle is considered totally separate from all others.

This is the best known use for duplicate file handles. Numerous articles have pointed out that the most efficient way to both flush buffers and update the disk directory with the current file length is to duplicate the file handle and then close the file using the duplicate handle. This is because closing a file usually takes less time than opening one. Opening a file requires that the file be located, possibly a lengthy process if an extensive directory structure must be navigated before the first read(s) are done to fill the buffer. This may seem odd to many programmers, especially if they have worked on systems where there is control on this sort of thing. Remember, though, that DOS has no control on how many references to a file are going on at the same time. After all, it took an entire new product (networking) to add capabilities like this to DOS.

Can you determine the file handle from a FILE pointer?

Yes, but this is *very* implementation dependent and should not be considered a portable coding practice.

In Microsoft 3.00 C the FILE stream pointer is a pointer to an array entry. Stdin is already defined in a header file to point to the first array entry, the one with subscript zero. Also, a property of C is that the difference between two pointers into the same array yields the number of elements between the entries that correspond to the pointers. Therefore,

```
handle = file_pointer - stdin;
```

will yield an integer that reflects the subscript value of the FILE pointer file_pointer. In the same way the file handle can act as a subscript into the array of structures of type FILE. Again, this is an implementation dependent arrangement.

Note: In general, FILE pointers and file handles should not be used to access the same file at the same time. The presence of buffering associated with the FILE pointer can cause some *very* unexpected results.

Can you do low-level I/O on a file after opening the file with a high-level open?

Yes, but if the high-level FILE access is buffered, the first "read" against the file will read a full buffer. If a low-level read is done, the characters *past* where the high-level access stopped will be returned. If the file is shorter than the size of the buffer, the end of the file will be sensed. Either set the buffer size to zero or rewind the file before doing the low-level access.

Again, mixing these approaches to I/O can yield unexpected results.

In Conclusion

Function 45H, then, is used to duplicate ("clone") the file handle of an existing file. This allows *all* work on the file to be accomplished through *either* of the handles. It also allows the characteristics of the file to be saved so that the particular handle that was cloned can be used to reference another file. This is necessary if some child process is counting on using that particular handle, but on using it with another file. If a programmer considers certain handles as "standard," he or she *will* need that particular one. By cloning the handle we can restore its characteristics after the child is done.

Function 46H is used to force the change in the file to which a particular handle refers. We provide a handle that already refers to a file and also the filename of another file. After 46H is done, the existing handle now refers to the new file, and the file to which the handle first referred is closed. The name of this function can be interpreted *not* as "force the duplication of a file handle," but as "force the duplicate file handle to refer to another file" although the function works with any handle and not just those returned by function 45H.

So this is how a parent process can redirect standard input for a child. First we use function 45H to clone the standard input file handle so that it won't be lost. Then we use function 46H to force the standard input file handle to reference the desired file. Finally, when the child is done, we use 46H again to force the standard input file handle back again to the cloned handle that we created at the beginning. Therefore, accomplishing redirection requires using at a minimum function 46H. We also need function 45H if we intend to restore things to the way they were after the process using the redirection is done.

Well, we haven't exactly proved mathematically that either standard file handles or redirection is required or necessary for programming. But some light has been shed on both their utility and the appropriate place where they can be found in "typical" PC systems. I, as much as anyone, am very interested in other ways that either of these concepts has or can be used. Let's hear from you experts out there. Give us beginners some tips. Thanks. □

References:

Jump, Dennis. *Programmer's Guide to MS DOS for the IBM PC*, 134-136. Brady Press, 1984.

Kernighan, Brian W. and P.J. Plauger. *Software Tools*. Addison-Wesley, 1976.

Pearson, Graham. "DOS 2 File Handles and File I/O." *Programmer's Journal*, July/August 1986: 24.

Microsoft MS-DOS Operating System Programmer's Reference. Part No. 036-014-003, Document No. 8411-200-01. 1984.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Terminate and Stay Resident

by M. Steven Baker

A DOS memory resident program can extend the functionality of DOS and provide some limited multi-tasking and context switching features in an otherwise single-user operating system.

This article is about the programmer's concerns in implementing a memory resident program and about the help provided by the Terminate and Stay Resident (TSR) features available in MS DOS.

When I first conceived this article, I thought to myself how straightforward a topic it should be. Just review the available DOS and IBM technical reference manuals, maybe dig through some old magazine articles and notes on memory resident programs, do a little snooping into DOS itself, SideKick, and a few other memory resident utilities, and review some TSRs I've written myself for clients. Unfortunately, the software waters on memory residency under DOS are good and murky.

Background Perspective

A Terminate and Stay Resident interrupt was incorporated in the earliest versions of MS DOS. Seattle Computer Products' 86-DOS Version 0.3 had this feature a year or two before Microsoft bought it. The expressed purpose of this operating system function call was to allow extensions to DOS, particularly user-written interrupt handlers, to be easily added to the operating system. MS DOS before Version 2.0 did not have the option of user installable device drivers, but TSR programs provided a somewhat equivalent feature. For example, early multi-function cards by AST™ and other manufacturers would come with TSR programs for the realtime clock, ram disk, and print spoolers.

With the introduction of MS DOS 2.0 by Microsoft, several TSR utility programs were included with the operating system for the IBM PC. These included PRINT.com, a memory resident print spooler of very limited features, and two simpler TSRs: GRAPHICS.com, a memory resident addition to the PRINT SCREEN function allowing screen dumps to an IBM graphics printer when in one of the three possible graphics video modes; and ASSIGN.com, a memory resident utility to assign logical disk drives to actual drives in a system. As with other aspects of DOS, Microsoft provided only limited details on the internal operations of MS DOS and no documentation on the 8086 interrupts and DOS function calls that support TSR system features. The background print spooler did demonstrate the possibility of limited multi-tasking under DOS, even though DOS was/is still a single-user operating system.

Meanwhile, back at the computer terminal, software developers large and small were introducing a whole raft of memory resident utilities for the IBM PC. There were the keyboard enhancers and macro programs (ProKey, SuperKey, etc.) and the popup notepad and phone utilities (SideKick, Spotlight, etc). These were followed by a dazzling variety of more exotic specialty programs including spelling checkers, thesauruses, outline processors, and graphics and database programs (Turbo Lightning, Word Finder, Ready, Graph-in-a-Box, etc.). At the same time, Microsoft expressed a general policy of not supporting the concept of TSR programs. Why incorporate TSR features and not document and support them? Because TSRs conflict with the long term goal of making DOS into a true multi-tasking/multi-user operating system. Most TSR programs are not "well behaved" and may take over some system hardware feature making true multi-tasking and multi-user operation difficult if not impossible.

Types of Memory Resident Programs

Let's step back a second and try to categorize the current flood of TSR programs. First, there are the simplest extensions to DOS itself, such as GRAPHICS.com and the various RAMDISK utilities. These programs extend a hardware feature on the IBM PC and do not need to use DOS function calls. Other examples of this first type would include the various SETCLOCK.com programs that replace the DOS time function calls with calls to a realtime clock. The DOS ASSIGN.com utility is a similar TSR that maps MS DOS disk calls from logical to actual disk drives in the system. The distinguishing feature is that once resident, the TSR makes no DOS function calls.

The second class includes the more complex TSRs that once resident must make DOS function calls (e.g., for file reading and writing). For my own understanding, I separate these TSRs into two types: (1) DOS extension programs and (2) limited multi-tasking and application programs. In the first category, I would place keyboard macro programs that allow reading and writing keyboard macro files "on the fly." The second type includes most of the specialty programs that make DOS function calls and provide some limited multi-tasking or task switching features. These programs may extend DOS with a printer spooler like PRINT.com, but more likely they provide some application program very conveniently at the touch of a "hot key."

Well Behaved Memory Resident Programs

The main concerns in writing a TSR are to create a "well behaved" program and, nowadays, make certain that other TSRs already loaded are also allowed to function. The simpler class of TSRs (such as GRAPHICS.com) are much easier to write than the second class (such as PRINT.com). Again, the distinguishing feature is whether DOS function calls must be made from the TSR. The difference is reflected in the size of these sample TSRs provided with DOS 2.0: GRAPHICS.com is 789 bytes, ASSIGN.com is 896 bytes, and PRINT.com is 4,608 bytes.

The difficulty with TSRs like PRINT.com, SideKick, Ready, etc., is in determining when DOS or an application program can be interrupted. This problem exists because DOS function calls are not re-entrant or recursive. If a DOS function call is in progress and our TSR interrupts and makes another DOS call, the first call will be trashed and lost with unpleasant side

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

effects for the primary application. Nor is the ROM BIOS Diskette_IO interrupt (int 13h) re-entrant. For example, if a disk read were started by seeking to a particular track and we interrupted before the reading took place and invoked a DOS call requiring the disk interrupt, the first seek would be lost upon re-entering the Diskette_IO 13h routine.

MS DOS provides two undocumented functions that support the TSRs that come with with the operating system. From my reading I gather that most authors of TSRs have uncovered these DOS features by disassembling DOS itself and PRINT.com—not from information willingly provided by Microsoft. This is how I "discovered" these hidden features.

Undocumented DOS TSR Support

The first undocumented DOS function call returns a byte pointer to a flag in DOS itself. I have seen this function referred to as InDOS and also as DOS_CRITICAL in a few articles. If this byte flag is 0, DOS is not currently active and therefore any DOS call can be made without trashing an active DOS call. This flag is not just a logical flag, but represents a count of recursive calls into DOS. This function cannot be called at any time since it goes through the DOS function entry and stack switch routine with interrupts enabled, allowing interruption. A TSR program when it is first loaded by COMMAND.com during initialization will make an INT 21 function 34 call and save the pointer for later use.

The DOS_CRITICAL or INDOS Function:

```
Entry:    AH = 34
          int  21h
Returns:  in ES:BX a pointer to a DOS_CRITICAL flag
```

The second undocumented DOS function is the Background Process function, INT 28h. There is one time that MS DOS can safely be interrupted even when DOS has been called (i.e., the DOS_CRITICAL flag is non-zero). If DOS is waiting for keyboard input, the DOS_CRITICAL flag is set to 1, but DOS calls INT 28h continually. The MS DOS PRINT utility appears to be the only DOS program that currently uses this interrupt, which would imply that its express purpose is to allow background utilities like spoolers to function. If a TSR program replaces INT 28h and monitors for calls to it, DOS functions above Print_String (function 9) may safely be used.

Documented TSR Support

MS DOS provides documented support of three functions relating to TSRs. The original function is INT 27h, the Terminate but Stay Resident function. This function is the traditional method for DOS programs to remain resident upon termination. Note that this function restores INT 22h, 23h, and 24h in the same way as a normal termination, so it cannot be used to install permanently resident Ctrl-Break or Critical Error Handler routines. The maximum size of memory that can be made resident by this method is 64K (actually about 63.9K since DX cannot be above 0FFF0h). Open files are not automatically closed by this function.

Terminate but Stay Resident: INT 27h

```
Entry:    CS = segment of Program Segment Prefix
          DX = offset of last byte + 1 (relative to Program Segment
               Prefix) to be remain resident
Returns:  nothing
```

DOS 2.0 and above added another equivalent (from Microsoft's perspective, the preferred function), Keep Process call, INT 21h Function 31h. This function allows the return of an exit code to the calling process and allows for larger (greater than 64K) resident code.

Keep Process (INT 21h) Function 31h

```
Entry:    AH = 31h        (DOS Function)
          AL = return code
          DX = memory size to reserve (in paragraphs)
```

The least used function is the Multiplex Interrupt (INT 2Fh)—also called the Print Spool Control function in some DOS documentation—which may be used for interprocess communication. PRINT uses this interrupt to pass information to a PRINT spooler already resident in memory. However, it may also be used by other processes. Every multiplex interrupt handler is assigned a specific multiplex number. The multiplex number is specified in the AH register. The specific function requested is specified in the AL register. The multiplex numbers AH = 0 through AH = 7Fh are reserved for DOS, but others are available for application program use. Function 0 (Get Installed State) is currently the only function that must be defined uniformly by all INT 2Fh handlers.

This interrupt was not documented until DOS 3.x versions of the *IBM DOS Technical Reference Manual* although it was used by the DOS PRINT Utility from DOS 2.0 onward. The DOS 2.0 PRINT utility makes calls with AH = 0 and AH = FFh, which are not documented, as well as with the documented calls. *IBM DOS Technical Reference 3.1* describes the Print Spooler Multiplex (AH = 1), the resident part of PRINT.com, in detail. *DOS Tech Ref 3.2* adds two more pre-defined DOS multiplex handlers: Assign (AH = 2 is the resident part of DOS 3.x ASSIGN) and Share (AH = 10h is the resident part of SHARE). However, the earlier ASSIGN command in DOS 2.x did not use this interrupt handler.

Multiplex Interrupt INT 2Fh

```
Entry:      AH = multiplex number
            AL = function code
            other registers as needed
Returns:    AX = error code if unsuccesful (carry set)
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Function 0 Get Installed State (must be supported)

```
Entry:          AH = multiplex number
                AL = 0
Returns:        AL = status (0, 1, or FFh)
                AL = 0    Not installed, okay to install
                AL = 1    Not installed, not okay to install
                AL = FFh  Already installed
```

Print Spool Multiplex Handler

```
Entry:     AH = 1    Resident Part of PRINT.com
           AL = 0    get print spooler installed status
                1    submit a file to be printed
                2    remove a file from the print queue
                3    cancel all files in queue
                4    hold print jobs for status read
                5    end hold for status read
           DS:DX segment:offset of packet address if function 1
                 segment:offset of ASCIIZ file specification if
                     function 2 (remove file)
Returns:   carry clear if successful
           For Function 0
           AL = status (0, 1 or FFh) as defined above

           For Function 4
                DX = error count
                DS:SI pointer to print queue
Returns:   carry set on error
           AX = error code
```

For function 1, the packet is 5 bytes long, the first byte contains the level, and the next four bytes contain a dword pointer of an ASCIIZ file specification (filename cannot contain wildcards). For function 2, wildcards (* and ?) are permitted in the filespec, allowing multiple files to be deleted. For function 4, the pointer returned for the print queue points to a linked list of ASCIIZ strings. Each entry in the queue is 64 bytes long, and the last slot has a zero first byte.

BUSINESS PROGRAMMERS

TURBO BUSINESS ACCOUNTING MODULES

CREATE CUSTOMIZED VERTICAL ACCOUNTING MARKETS

FIVE SEPARATE MODULES
- General Ledger
- Inventory
- Accounts Payables
- Cost Accounting
- Accounts Receivables

STANDARD WITH EACH MODULE
- Full Documentation
- Executable File
- Documented Pretty Printed Source Code

ALL INTEGRATABLE! NOT COPY PROTECTED NO ROYALTY FEES

All for the price of $49.95 per module. Get three modules for the price of two at $98.95. Or the complete set which includes all five for the introductory low price of $199.95. Prices are valid until January 1, 1987.

(Note: Each module comes on a DSDD 5¼ disk)

SYSTEM REQUIREMENTS: All of the above modules require TURBO PASCAL 3.0 or BETTER, TURBO DATABASE TOOLBOX, 256K, and an IBM PC/XT/AT or MS-DOS Compatible.

Call or write for full information.

PS BUSINESS SYSTEMS
4400 Wallingford Ave. N. Suite 3
Seattle, WA 98103 U.S.A.
(206) 632-4568
COMPUSERVE 73267,1513

CHECK, MONEY ORDER OR COD IN U.S. FUNDS

SHIPPING: $5.00 U.S., $7.50 CANADA, $10.00 Caribbean, Hawaii $20.00 Overseas Air.

Turbo Pascal and Turbo Database Toolbox are registered trademarks of Borland International, Inc.; MS-DOS are trademarks of Microsoft Corporation; IMB, PC, XT, and AT are registered trademarks of International Business Machine Corp.

A Simple Memory Resident Program

The DOS GRAPHICS utility is an example of a very simple TSR program. Upon invocation, GRAPHICS tests to make sure that it has not been previously installed: It gets the current INT 5 Print Screen vector and compares a number of bytes of the current handler with GRAPHICS. Once we know that GRAPHICS isn't already there, a new INT 5 vector is installed pointing to GRAPHICS resident code. This program also goes that extra mile, giving an example of byte saving for small memory machines. DOS loads COM programs at 100h, and the area in the base page from 0 to 5ch must be preserved for termination. GRAPHICS goes to the trouble of relocating the resident interrupt handler down to 60h in the base page to save 160 bytes of memory.

The ASSIGN Command—a Little More Complexity

The DOS ASSIGN utility is an example of a more complex TSR program. It modifies DOS itself but does not make any DOS calls when resident. This utility allows the user to "assign" any logical drive (from A-H in DOS 2.0) to an actual physical drive. Its main purpose is to tolerate those poorly written programs that absolutely expect their data files to be in Drive "B" to run on a hard disk machine. ASSIGN does this logical-to-actual drive mapping by replacing the following interrupts:

INT 21h Dosint—patches the filespec and disk drive forwarded on to the original DOS interrupt. This is quite complex given the variety of DOS functions and ways that disk drives may be included.

INT 25h Absolute Disk Read—patches drive number in AL, then forwards to original interrupt.

INT 26h Absolute Disk Write—patches drive number in AL, then forwards to original interrupt.

The Diskette_IO INT 13h is not modified, so calls to the ROM BIOS will not be affected by this utility. The memory resident part of ASSIGN maintains a simple table of 8 bytes that map to actual disk drives. When a drive is assigned, the value in this table is changed to reflect the actual drive to be used as this logical disk. When the ASSIGN command is given without parameters, the default table is copied into the memory resident code.

```
disktabl  db   1,2,3,4,5,6,7,8  ;default disk assign table,
                                 ;ie, logical drive A = disk 1
```

A Model Memory Resident Program

We can look at the DOS PRINT utility as a model TSR program. It is a good example because it uses both the documented and undocumented TSR functions in DOS. After it is memory resident, it must also make DOS calls for reading disk files while background printing. Two hooks are used for background printing: the Background INT 28h and the Timer_Tick INT 1Ch. These two interrupts are vectored to PRINT, which then tests several other monitors to make certain that DOS can be interrupted. The following is a rough outline of the initialization routine for a TSR similar to that used in the DOS PRINT utility which must make DOS calls:

1. Test for DOS version above 2.0 since we'll be using some undocumented features; if not okay, exit.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

2. For the PRINT spooler, open the appropriate list device to make certain that it exists.
3. Save old Background INT 28h, and install our own.
4. Save old Multiplex INT 2FH, and install our own.
5. Save old Diskette_IO INT 13h, and install our own.
6. Save existing Printer_IO (INT 17h), Rs232_IO (INT 14h), and Print_Screen (INT 5) interrupt vectors.
7. Depending on selected print device, install new interrupt vector:

```
If PRN:, LPT1:, LPT2, LPT3 install new Printer_IO (INT 17h) and
   Print_Screen (INT 5) vectors
If COM1:, COM2:, COM3 install new RS232_IO (INT 14h)
```

8. Call DOS_CRITICAL function (INT21h function AH = 34h) and save pointer.
9. Save old Timer_Tick INT 1Ch and install new vector.

When the resident part of PRINT is invoked by one of the two hooks, it checks to make certain that INT 13h (Diskette_IO) was not in progress, and if entry was by INT 1Ch (Timer_Tick), it also checks the DOS_CRITICAL flag. PRINT is also careful about saving and restoring certain other parameters before invoking DOS calls that might return an error such as reading a file. The resident part of PRINT has a 512-byte buffer for reading files to spool from disk. The following is a rough outline of what is done before and after a disk read to protect an application program:

1. Save old DTA (disk transfer address).
2. Save old INT 24h Critical Error Handler.
3. Install our own INT 24h vector (Error Handler) to catch any errors.
4. Make DOS call for disk read.
5. Restore old INT 24h Critical Error Handler.
6. Restore old DTA (Disk transfer address).

Programming Guidelines for TSR Programs

Microsoft has provided limited distribution on a February 1986, draft specification for TSR programs. These draft guidelines define a set of operational rules for TSR programs to minimize conflict when other memory resident utilities are loaded. They do *not* document or describe the "undocumented" features of DOS that TSR implementors need to be aware of. The gist of the suggestions is "mimic DOS and the ROM BIOS as much as possible." Some valuable basic guidelines help in the following areas:

General Issues:
Do not set a timer trap to re-grab an interrupt vector that you initially installed (e.g., SideKick). It should not be critical that your program be the last vector in the chain.

Setting up a local stack in an interrupt routine prevents it from being re-entrant. Set up and use a local stack only when needed and hardware interrupts are off, in an area protected from re-entrancy.

Do not jump from a DOS critical error handler. Instead, set a global error flag that your device routines check, issue an IGNORE response, and IRET to DOS. This assures that the DOS stacks will be cleared properly.

Keyboard Issues:
Don't take complete control of INT 9 (hardware keyboard interrupt). If your program wants to use keystrokes that the PC ROM BIOS doesn't generate (e.g., the Ctrl-Up arrow omission), install your routine, chain into the interrupt, generate a new scan code, and put it into the PC's keyboard buffer just as the ROM BIOS does. If you are looking for a single keystroke, set a flag and continue through the interrupt. When the TSR is running, it should use INT 16h (Keyboard_IO) to get a keystroke and should not just call the installed interrupt vector. This allows all TSRs in the chain to see the key request and take any appropriate action.

Video Issues:
Use the ROM BIOS whenever possible. If you directly change video modes, cursor types, background intensity, EGA palette values, etc., save the new values in the appropriate ROM BIOS and EGA data areas.

This draft guideline also proposes an extensive format of data records for TSR programs and a program interface. Each installed TSR interrupt vector would be preceded by a six-byte record that would point to an Interrupt INformation Record (IINR) that contains more information on this interrupt procedure. In turn, these IINRs would point back to other data records describing details of the TSR.

The program interface proposed would use Cassette_IO INT 15h (function AH = 52h) as a method for the non-resident parts of programs to communicate with resident parts.

Continued...

FORTRAN PROGRAMMERS

Looking for the right PC FORTRAN LANGUAGE SYSTEM? If you're serious about your FORTRAN programming then you should be using F77L- LAHEY FORTRAN.

Editor's Choice - PC Magazine

- Full FORTRAN 77 Standard (F77L is not a subset)
- Popular Extensions for easy porting of minicomputer and mainframe applications
- COMPLEX*16, LOGICAL*1 and INTEGER*2
- Recursion - allocates local variables on the stack
- IEEE - Standard Floating Point Arithmetic
- IMPLICIT NONE
- Long variable names - 31 characters
- Fast Compile - Increase your productivity
- Source on Line Debugger (Advanced features without recompiling)
- Arrays and Common Blocks greater than 64K
- Clear and Precise English Diagnostics
- Compatibility with Popular 3rd Party Software (i.e. Lattice C)
- Easy to use manual
- Technical Support from LCS

F77L - THE PROGRAMMER'S FORTRAN
$477.00 U.S.

System Requirements: MS-DOS or PC-DOS, 256K, math coprocessor (8087/80287)

FOR MORE INFORMATION: (702) 831-2500

Lahey Computer Systems Inc.
P.O. Box 6091 Incline Village, NV 89450/U.S.A.

International Dealers:

England:	Grey Matter Ltd.,	Tel: (0364) 53499
Denmark:	Ravenholm Computing,	Tel: (02) 887249
Australia:	Computer Transitions	Tel: (03) 537-2786
Japan:	Microsoftware, Inc.,	Tel: (03) 813-8222

SERVING THE FORTRAN COMMUNITY SINCE 1967

Circle Number 44 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

PRODUCTIVITY TOOLS

From Opt-Tech Data Processing

NEW Version

Opt-Tech Sort™

NEW Version

ALL NEW Version 3.0 features even faster sorting, record selection, output record reformatting, dBASE III files, comma delimited fields, and much more. This high performance sort/merge/record selection utility can be used as a stand-alone program or called as a subroutine from most languages.

Supports unlimited filesizes, multiple input files and fixed or variable length records. Many special file types are supported including Btrieve and dBASE. Up to nine sort control fields (ascending or descending), all common data types supported. Output files can be combinations of full records, keys or pointers, subsets of the input file fields, and literal values.

Written in assembly language for **high performance.** Example: 4,000 128 byte records sorted to give key and pointer in 30 seconds. **$149.**

On-Line Help™

A comprehensive utility for adding help windows to your programs. It provides efficient utilities and routines for interfacing your programs with the help system routines and help message libraries.

Help windows are displayed in a fraction of a second. You have total control over the contents of the window, its size and its position on the screen, including the display and border colors.

On-Line Help can be interfaced with interpreted Basic and all popular compilers. **$149.**

Scroll & Recall™

Allows you to conveniently scroll back through data that has gone off the top of your display screen. Up to 27 screens of data can be recalled or written to a disk file (great for documenting systems operations).

Allows you to easily recall and edit your previously entered DOS commands and data lines without re-typing.

Scroll & Recall is very easy to use. It's a resident utility that's always there when you need it! **$69.**

All programs IBM PC/XT/AT & MS-DOS compatible.

Visa, M/C, AMEX, Check, Money Order, COD or Purchase Orders accepted.

Quantity and Dealer Discounts Available

To order or to receive additional information write or call:

Opt-Tech Data Processing

P.O. Box 678 - Zephyr Cove, NV 89448

(702) 588-3737

Circle Number **16** on the Reader Service Card

Inquiries regarding this document should be addressed to

Adrian King, Director
OS Product Marketing
Microsoft Corp.
16011 NE 36th St.
Redmond, WA 98052
(206) 828-8080

The Bottom Line

As a programmer, do I use memory resident programs myself? No, not normally except for special situations. I want as little to interfere with my programming and testing as possible. Oh, I wouldn't mind a good screen capture utility now and then. But the bulk of users do appear to like TSRs and that drives programming services. From my perspective, TSRs do serve a useful purpose when modifying hardware or ROM BIOS routines. I personally prefer to use "program loaders," small programs that load another application and for that application change the DOS or hardware environment. Then, when the application is done, they disappear and go away. Maybe the kernel for another article? □

Some References for Further Reading:

IBM Technical Reference Manual. Publication #1502234, revised April 1983.
The hardware reference to the IBM PC. Section 2 contains a complete listing of the ROM BIOS essential for most serious PC work. This is the most heavily used publication on the IBM PC in my library. A version specific to the IBM AT is also available (Publication #636166), but mostly I still use the PC version.

IBM Disk Operating System Technical Reference. Publication #6139656, March 1986.
Technical information for a specific version of DOS, the official line. The latest version is for DOS 3.2.

"Best of BIX." *Byte*, September 1986.
Byte's on-line conference for the IBM PC has carried discussions on the PRINT.com program. The Pascal conference has several discussions by staff from Living Videotext (Ready) on TSR programs and technology used by Ready. Borland's SIG on CompuServe also has an active group of programmers studying TSR software.

Duncan, Ray. *Advanced MS DOS*. Microsoft Press, 1986.
The officially sanctioned guide to DOS for assembly language and C programmers.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Experience Prolog

with

APT™

Active Prolog Tutor

Learn Prolog While Building Applications Interactively

A P T is *designed* to teach you. With the guidance of the text and the support of the Prolog interpreter, you can experiment with new concepts as they are presented.

Use a Special Prolog Interpreter

Prolog concepts are shown in text form and then demonstrated by example using the Prolog interpreter. You can also use the interpreter as a practice environment to try ideas and get immediate feedback. A single keystroke toggles you between the tutorial text and the practice interpreter. You can write a program and execute it without leaving the tutorial environment.

The practice interpreter is designed for education and is integrated with the tutorial package. The interpreter is not appropriate for production applications.

See for Yourself How Prolog Works

Traditional teaching tools have been ineffective in presenting what happens as Prolog executes. You need to know *how* Prolog behaves in order to write useful programs. A P T has 3 tracing mechanisms which will literally **show** you:

--the execution pattern of Prolog, and clause order importance.
--how values are passed up and down in a recursive Prolog routine.
--how a Prolog query is processed.

System Requirements
A.P.T. requires an IBM PC, XT, AT or compatible hardware and PCDOS 2.0 with a minimum 512K

Investigate Several Applications

A P T is a self-contained practice environment. It guides you through the development of:

- an adventure game
- an intelligent genealogical database
- an order entry system
- an identification expert system
- a natural language user interface

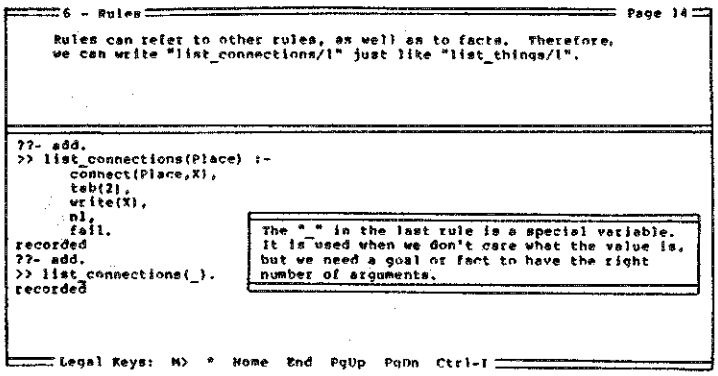
Each program builds on the one before. By the end of the tutorial, you have actually built working Prolog applications.

"APT is fun to use. It is very highly interactive. At almost anytime, you can terminate the formal learning process and play around. This is THE introduction to Prolog."

— Larry Lidsky
Professor of Nuclear Engineering, MIT

All the Extras You Need

- Manipulate the tutorial with single keystrokes
- Move to prior pages easily
- Instantly access any chapter with the table of contents
- Helpful hints appear in pop-up windows
- Keys you can use are displayed at the bottom of the screen
- Automatically begin where you left off
- All work is automatically saved when you leave the tutorial
- The Prolog syntax you learn is compatible with:
  --The Clocksin & Mellish standard
  --Prolog 86 and Prolog 86 Plus by Solution Systems
  --Arity Standard Prolog and Arity Compiler/Interpreter
  --Also compatible with Turbo Prolog's version of Prolog syntax


6 - Rules
Page 14
Rules can refer to other rules, as well as to facts. Therefore, we can write "list_connections/1" just like "list_things/1".
??- add.
>> list_connections(Place) :-
connect(Place,X),
tab(2),
write(X),
nl,
fail.
recorded
??- add.
>> list_connections(_).
recorded
The "_" in the last rule is a special variable. It is used when we don't care what the value is, but we need a goal or fact to have the right number of arguments.
Legal Keys: N> * Home End PgUp PgDn Ctrl-I

A common APT screen will include a tutorial description of what is going on (top window), interpreter code (bottom window), and a pop-up window with helpful hints. Legal keys are always displayed on the bottom of the screen.

Solution Systems™

MONEY-BACK GUARANTEE
Try Active Prolog Tutor ($65) for 30 days — if not satisfied get a full refund.
TO ORDER CALL (800-821-2492)

SOLUTION SYSTEMS, 335-J WASHINGTON ST., NORWELL, MA 02061. 617-659-1571

Circle Number 47 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Turbo Pascal Inline Code

by James M. Smith

I recently spent two years writing a large project in Turbo Pascal (23,000 lines, 700K of source code, 160K of object code).

Writing in Turbo has some obvious advantages over writing in Assembler—such as ease of prototyping and of debugging. On the other hand, although Turbo Pascal code is as fast and compact as the code from even many multipass, optimizing compilers, it is still two, three, four, and even more times larger and slower than equivalent hand-assembled code.

Inline Code

As the Turbo project began to reach maturity, the amount of debugging and prototyping began to diminish, and it made sense to convert the time-critical code to Assembler. I used the perhaps under-utilized Turbo feature, inline code.

The shell has remained in Turbo for those times (always) when further prototyping and debugging are needed.

What about Lost Portability with Inline Code?

I say forget portability. I should be so lucky as to write something that is even worth porting. I watched a company with millions go broke, partly at least by trying to stay portable and ending up with a slow product.

Besides, I've done the work first in the high level language and have retained the high level code as comments.

Here's the process I've developed for changing to inline code. Included is a program that manages Debug disassembly, a shell for writing Assembler for Turbo, and a program which converts Microsoft's MASM .LST output to Turbo inline code.

First Step: Finding the Compiler's Output

The first step is to isolate the compiler output. I sandwich these two inline statements around Pascal code of interest:

```
inline($90/$90/$90);

inline($90/$90/$90);
```

Hex 90 is the Assembler instruction NOP, or no operation. Compile the Turbo code to a .COM file, then go in with Debug and search for 90 90 90. That sequence doesn't normally occur in a Turbo program.

If you haven't used Debug before: You invoke Debug with your program name on the command line. When Debug loads your program and gives you its dash prompt, type r to look at the registers. CX holds the length of your file. If CX is AB5F, type s 0100 lAB5F 90 90 90 (which means: Search from offset 0100, length AB5F, for 90 90 90).

Here's a piece of source code to be optimized:

```
var current_status, last_status: integer;

 begin
  if current_status<240 then
      if current_status<>last_status then
          last_status:=current_status;
 end.
```

Disassembling

After this code was compiled to .COM and the code address was located, I used Debug's Unassemble command, and the disassembly was moved into a textfile for access by a word processor.

To automate disassembly, you can use the accompanying utility SOURCE.PAS, which takes as parameters the starting and stopping addresses of the code you want and produces a textfile named 'Z,' which can then be a redirection input file for Debug. First, do

```
SOURCE 2D9F 2DBF
```

(or whatever the addresses are). Then

```
DEBUG YOURPROG  <Z  >YOURFILE
```

Debug's disassembled output ends up in the file YOURFILE. Debug disassembled the Turbo code above like this:

```
                                          (comments added)
0CF6:2DA2 A16202    MOV   AX,[0262]   (load current_status into AX)
0CF6:2DA5 3DF000    CMP   AX,00F0     (compare to 240)
0CF6:2DA8 7C03      JL    2DAD        (jump if less to 2DAD)
0CF6:2DAA E91200    JMP   2DBF        (jump out)
0CF6:2DAD A16202    MOV   AX,[0262]   (load current_status into AX)
0CF6:2DB0 3B066002  CMP   AX,[0260]   (compare to last_status)
0CF6:2DB4 7503      JNZ   2DB9        (if not equal, jump to 2DB9)
0CF6:2DB6 E90600    JMP   2DBF        (jump out)
0CF6:2DB9 A16202    MOV   AX,[0262]   (load current_status into AX)
0CF6:2DBC A36002    MOV   [0260],AX   (move AX into last_status)
```

The first figures on each line show the starting segment and offset of that line's code. Next is the code itself. Then comes the line's Assembler command.

You can puzzle out the meaning of the disassembly by referring to the Pascal code.

```
MOV AX,[0262]
```

means: Move the integer at location 0262 in the data segment into the AX register. The subsequent comparison of AX with Hex F0 (240) lets us know that 0262 is the location of the variable "current_status." Thus, "last_status" must be the variable at location 0260.

A word about data locations: Turbo uses a single 64K segment for program data. A variable is referred to by its offset from the start of the segment.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

This example shows some common opportunities for optimizing: Code produced by the compiler will load registers even though they already contain the correct memory value, and jumps are unnecessarily complicated.

Optimizing

Next, I bring the textfile into a word processor to optimize it. I use *The 8086 Book* by Russell Rector, McGraw-Hill, as a reference.

I optimized the code thus:

```
  MOV       AX,local_word   ;current_status
  CMP       AX,00F0h
  JAE SHORT out
  CMP       AX,local_word   ;last_status
  JZ  SHORT out
  MOV       local_word,AX   ;last_status
out:
```

Note that the Debug disassembly of the number 00F0 has to be changed to the format 00F0h before Microsoft's MASM can be used. (A leading alphabetic hex digit must also always be preceded by a "0.")

The Turbo variables are referred to in the Assembler listing by the dummies "local__word" and "local__byte." Put the actual variable names in the Assembler comment field. After the listing is assembled, NATIVE.PAS will take the .LST output of the assembler (which includes the comment field as well as the code) and convert it to inline. The variable names in the comment fields will be included in the inline as Pascal comments, and they will guide you in moving the names into the inline code.

Assembling

Next, the code is put into SHELL.ASM for assembly. This shell begins and ends with the proper Assembler directives to generate code compatible with Turbo.

When you assemble SHELL.ASM, send the Assembler output to SHELL.OBJ and have the Assembler produce SHELL.LST.

Converting to Inline

The Turbo program NATIVE.PAS, included in this article, takes SHELL.LST and turns it into the file RESULT.PAS, which looks like this:

```
inline(
$A1/$00/$00/            {  MOV    AX,local_word   ;current_status  }
$3D/$00/$F0/            {  CMP    AX,00F0h                         }
$73/$09/                {  JAE SHORT out                           }
$3B/$06/$00/$00/        {  CMP    AX,local_word   ;last_status     }
$74/$03/                {  JZ  SHORT out                           }
$A3/$00/$00/            {  MOV    local_word,AX   ;last_status     }
                        {out:                                      }
      );
```

With a little cleanup, including substituting variable names in the inline code, the listing looks like this:

```
inline(
$A1/current_status/     {  MOV    AX,local_word   ;current_status  }
$3D/$00/$F0/            {  CMP    AX,00F0h                         }
$73/$09/                {  JAE SHORT out                           }
$3B/$06/last_status/    {  CMP    AX,local_word   ;last_status     }
$74/$03/                {  JZ  SHORT out                           }
$A3/last_status         {  MOV    local_word,AX   ;last_status     }
                        {out:                                      }
      );
```

The compiler replaces the variable names with the variable offsets at compile time.

Accessing Data Structures with Inline Code

Arrays

Given:

```
var this: array[1..10] of integer;
    i,j,k: integer;
```

this:

```
j:=this[i];
```

looks like this:

```
MOV     AX,[0404]       (get i)
SHL     AX,1            (multiply by two to access integers)
XCHG    DI,AX           (move into DI)
MOV     AX,[DI+025E]    (get the array element)
MOV     [0406],AX       (move into j)
```

This code:

```
var
   that: array[1..10,1..20] of integer;
   i,j,k: integer;

   j:=that[i,k];
```

looks like this:

```
MOV     AX,[0404]       ;load k
MOV     CX,0028         ;multiply by 40 to access integers
MUL     CX
XCHG    DI,AX           ;move into DI
MOV     AX,[0408]       ;load i
SHL     AX,1            ;mult by 2 to access integers
ADD     DI,AX           ;add to DI
MOV     AX,[DI+024A]    ;DI plus offset of THAT
MOV     [0406],AX       ;move into j
```

This compiler code is tight and is presented here to show how an array address is generated. However, more complicated operations with an array, such as repeated reads and writes, present plenty of opportunities for optimization.

Continued...

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Pointers and the Heap

Pointers are used to access the heap. The compiler commonly uses the instruction,

```
LES     DI,DWORD PTR LOCAL_WORD
```

(load DI and ES with double word pointer starting at local_word) to load a pointer's value (two integers). Any code that uses pointers has potential for optimization because the compiler reloads DI and ES each time a pointer is mentioned. Consider the following:

```
type
    that = record
             it: integer;
             is: integer;
           end;

var this: ^that;   {a pointer to data type 'that'}
    i,j: integer;


i:=this^.it;
j:=this^.is;
```

The Debug disassembled compiler output looks like this:

```
LES     DI,[0260]       (load pointer 'this')
ES:
MOV     AX,[DI]         (mov 'it' into AX)
MOV     [0264],AX       (mov AX into 'i')
LES     DI,[0260]       (load pointer 'this')
ES:
MOV     AX,[DI+02]      (mov 'is' into AX)
MOV     [0266],AX       (mov AX into 'j')
```

When optimized and ready for MASM, it might look like this:

```
LES     DI,dword ptr local_word     (load pointer 'this')
MOV     AX,ES:[DI]                  (mov 'it' into AX)
MOV     local_word,AX               (mov AX into 'i')
MOV     AX,ES:[DI+02]               (mov 'is' into AX)
MOV     local_word,AX               (mov AX into 'j')
```

The command ES: tells the CPU to use the Extra Segment instead of DS (the default) for the following instruction. Note that Debug disassembles ES: with a syntax different from that required by MASM.

Access a record on the heap like this:

```
type
  this = record
           a: byte;
           b: byte;
           c: byte;
         end;


  MOV    AL,ES:[DI]      ;load a
  MOV    AL,ES:[DI+1]    ;load b
  MOV    AL,ES:[DI+2]    ;load c
```

Procedure Parameters

In a procedure or function, local variables and parameters passed on the stack are accessed with the BP register, whose default segment is SS.

Here's procedure code that accesses a variable passed on the stack. (It also illustrates access to I/O ports with the IN and OUT instruction. A port can be addressed directly if its address is below 256; otherwise, its address is loaded into DX and it is addressed indirectly):

```
procedure system(x:integer);
 begin
   while (port[$331] and 64)=64 do;   {looking for bit 6 to go low}
   port[$330]:=x;                     {send}
 end;
```

This compiles to 32 bytes of code. Optimized, it shrinks to 15 bytes and looks like this:

```
inline(
$BA/$31/$03/    { MOV     DX,0331       port address }
$B3/$40/        { MOV     BL,40         test value   }

                (here:)
$EC/            { IN      AL,DX         read port    }
$84/$C3/        { TEST    AL,BL                      }
$75/$FB/        { JNZ     here                       }

$8A/$46/$04/    { MOV     AL,[BP+04]    load 'x'     }
$4A/            { DEC     DX                         }
$EE             { OUT     DX,AL         write to port}
    );
```

The parameter is loaded from the stack at [BP + 4].

Here's why you should choose Periscope as your debugger...

You'll get your programs running fast. "It works great! A problem we had for three weeks was solved in three hours," writes Wade Clark of MPPi, Ltd.

You'll make your programs solid. David Nanian says, "I can't live without it!! BRIEF, a text editor my company wrote, would not be as stable as it is today without Periscope."

You'll protect your investment. We won't forget you after the sale. You'll get regular software updates, including a FREE first update and notice of later updates. You'll get technical help from Periscope's author. And you'll be able to upgrade to more powerful models of Periscope if you need to. One Periscope user writes, ". . . your support has won over even the heart of this hardened programmer!"

You deserve the best. Thousands of programmers *rely on the only debugger that PC Tech Journal* has ever selected as **Product of the Month** (1/86). You owe it to yourself to find out why, first hand.

You can try it at no risk. You get an unconditional 30-Day, Money-Back Guarantee, so you can't lose.

Start saving time and money now — order toll-free, 800/722-7006. Use MasterCard, Visa, COD, or a qualified company purchase order. As one user puts it, Periscope is "one of the rare products, worth every penny!"

Periscope I, software, manual, protected memory board and breakout switch $295
Periscope II, software, manual, and breakout switch $145
Periscope II-X, software and manual $115

Add shipping - $3 US; $8 Canada; $24 elsewhere. Ask about air shipment if you can't wait to get your programs up and running!

PERISCOPE
The Periscope Company, Inc.
(formerly Data Base Decisions)
14 Bonnie Lane, Atlanta, GA 30328 404/256-3860

Circle Number 6 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Calling Procedures and Functions

Turbo calls a procedure like this:

```
0CF6:2DD4 E8C8FF          CALL    2D9F
```

Note that the value in the code, FFC8, is not the same as the offset of the procedure, 2D9F.

The CPU adds FFC8 to the instruction pointer to generate the offset to the procedure. The compiler will replace a procedure or function name in inline code with its offset, but that is the wrong value to generate a call to the proper place.

An alternative method of calling a procedure that will work in inline code is to load the offset of the procedure into AX and then do a CALL AX.

The assembler instructions:

```
MOV   AX,8888h          ;load immediate value
CALL  AX
```

become this inline code, which calls the procedure "system":

```
inline(
$B8/system/          {      MOV  AX,system  }
$FF/$D0              {     CALL AX          }
      );
```

Also, use the included NATIVE.PAS code. It is more thoroughly commented.

Overlays

An overlay procedure isn't called directly—control is passed to code that first must determine whether the procedure must be loaded from disk. Thus, you can't call an overlay procedure directly from inline code. This is a good reason for adding procedure calls from inline code only when your program is in final shape and you can be sure what will be overlayed and what won't.

Passing Parameters on the Stack

Parameters are passed to a procedure on the stack. Thus, given:

```
var i: integer;
    k: byte;

procedure thing(a: integer, b:byte);
 begin
 end;
```

the call

```
thing(i,k);
```

looks like this:

```
0CF6:2DB8 A16002          MOV     AX,[0260]    ;get i
0CF6:2DBB 50              PUSH    AX
0CF6:2DBC A06202          MOV     AL,[0262]    ;get k
0CF6:2DBF 32E4            XOR     AH,AH
0CF6:2DC1 50              PUSH    AX
0CF6:2DC2 E8DAFF          CALL    2D9F
```

Note that the minimum size of something on the stack is a word. When a byte value is passed, the high order byte of AX is XORed to zero and the whole register is pushed.

Register Variables

It's possible to gain speed by passing variables to procedures in registers instead of on the stack. Stack manipulations with the 8086 are not particularly fast. You'll need to put corresponding inline code in the procedure to handle the register variables.

Continued . . .

If BASIC's data limit is the question . . .

DataSpace

. . . is the answer

DataSpace gives MS* and GW-BASIC* users . . .

- the ability to define and access string, integer, single, and double precision data arrays
- 10 - one, two or three-dimensional arrays up to 64K each - 640K total
- array storage and retrieval from files in any disc directory
- full compatibility with interpreters or compilers

. . . all with absolutely no impact on BASIC's own data space (requires MS-DOS* 2.10 or higher)

The only answer around for $75 . . .

DESIGN SOLUTIONS INC.

732 Sixth Avenue
Aurora, IL 60505
(312) 898-0651

* MS, GW-BASIC, and MS-DOS are trademarks of Microsoft Corporation

Circle Number 41 on the Reader Service Card

PL/D Compiler

. . . the fastest step from idea to code

Ultra-fast compile rate: PL/D (tm) is the new compiling standard at 6000+ lines/minute on an 8088 floppy system and is even faster with 286 and hard disc. No added link/assembly required.

Structured: PL/D supports nested FOR-NEXT, CHOICE-CASE-BUT-ENDCH, WHILE-THEN-ENDWHILE, SUBR-ENDSUB, and IF-THEN-ELSE-ENDIF structures at run-time and at compile-time. Symbols within SUBR are local.

Compile-time arithmetic: PL/D uses braces "{..}" to enclose a compile-time expression, force a variable to evaluate as an address, and access compile-time facilities such as a stack, symbol table, and code. Same capability as C "address-of" and more.

Extendable: PL/D has MACROs, the ability to include source libraries, and machine language definition for building extensions. In addition to statements, extensions can be elements of expressions because PL/D never re-orders expressions.

Code control: PL/D separates procedure and variable so that code may be placed in EPROM. You control placement during compilation, either over-lapping or not. Assignment of segment registers CS, DS, ES, and SS may be over-lapped or not. You can access all addresses, I/O devices, and instructions.

Integrated listing: PL/D indents to show nesting structure. Hex arithmetic and page flipping are not needed to find addresses during DEBUG because PL/D shows the actual address and last subroutine called for every line. Partial listings can be based on range, name occurrence, and included source files.

Source available: PL/D is written entirely in PL/D. Easy to modify. Learn about compilers. PL/D could spawn a mini-industry that includes you; act now.

only $49.95 + $75 for source. Add tax in CA.
NOT copy protected. MS-DOS 5 1/4" disk, 192 KB system.

DAIR Computer Systems VISA MASTER CARD
3440 Kenneth Dr., Palo Alto, CA 94303
(415) 494-7081

Circle Number 5 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Some Dispensable Procedure Code

Procedures always begin and end with this code, whether there are any local or passed variables that need to be referenced with BP or not:

```
0CF6:2DB0 55            PUSH    BP      {beginning}
0CF6:2DB1 8BEC          MOV     BP,SP
0CF6:2DB3 55            PUSH    BP
0CF6:2DB4 E90000        JMP     2DB7

0CF6:2DB7 90            NOP     (body of the procedure)
0CF6:2DB8 90            NOP     (body of the procedure)
0CF6:2DB9 90            NOP     (body of the procedure)

0CF6:2DBA E90000        JMP     2DBD    (end)
0CF6:2DBD 8BE5          MOV     SP,BP
0CF6:2DBF 5D            POP     BP
0CF6:2DC0 C3            RET
```

The JMP to 2DB7, which here jumps just to the following instruction, is used to make an area for storage of any constants declared in the procedure. Perhaps the JMP to 2DBD is for additional storage when 2DB7's jump capacity reaches its limit.

In any case, bypassing the jumps when there are no constants declared in a procedure has never produced an observable effect.

You can bypass the beginning code by using this call:

```
MOV AX,8888h    ;load immediate value
```

which is replaced with the procedure name +7 in the inline code:

```
$B8/system+7/   { MOV   AX,system+7     }
                { CALL  AX              }
```

This jumps 7 instruction bytes into the procedure.

At the end of the procedure, use your own inline RET instruction to bypass the end stack register manipulations.

This is getting down to the nitty end of procedure optimization—saving 13 instructions, or about 70 clocks. If you're to this point, moving the code into the caller and eliminating the procedure might be an option.

Interrupt Handlers

The BP code at the start and end of a procedure was the source of an early bug in Turbo for those using an interrupt handler. An interrupt handler looks like a normal procedure to Turbo (even though it is never called), so it gets the BP code at the beginning and end. An interrupt handler returns with an IRET, which bypasses the end BP code, but the starting BP code still executed, screwing up the stack. I remember Philippe Kahn sweating over this himself when he was doing the first support on the product. Borland remedied this by suggesting duplicating the ending BP code as inline code before your IRET. A faster way is to load the interrupt handler's offset +7 into the interrupt table so that the beginning code is just bypassed.

Setting the Data Segment in an Interrupt Handler

Your interrupt routine might be invoked at any time. If that happens during a DOS or BIOS call, the DS register won't necessarily be pointing at your program's data segment—and that's trouble if you try to access program data.

You need to store the DS register in the Turbo code segment because the code segment won't change. Declare it as

```
const
  data_segment: integer = 0;
```

Before you enable your interrupt routine, load the data_segment variable:

```
data_segment:=dseg;
```

Use code like this in the interrupt routine before you access data:

```
inline($1E);    {push ds}

inline( $2E/                    {assume cs}
        $A1/data_segment/       {mov ax, local word}
        $8E/$D8                 {mov ds, ax}
              );
```

At the end of the interrupt routine, be sure to pop the original DS:

```
inline(
$1F     { POP  DS }
      );
```

Some Benches

How fast is optimized code? Here's a bench of this article's first routine:

	Turbo Code	Optimized Inline	
Total Instructions:	29	17	Size Reduced 41%
Time for Maximum Path:	4.95	3.46	31% Faster
Time for Minimum Path:	2.47	2.2	11% Faster

□

Code follows.

Marcus struggles to understand the judges decision to order him to pay wife support, child support, and software support.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
program source;
var
   this,that:string[4];
   it: text;

begin
  if paramcount=2 then
    begin
      this:=paramstr(1);
      that:=paramstr(2);
      assign(it,'z');
      rewrite(it);
      writeln(it,'u ',this,' ',that);
      writeln(it,'q');
      close(it);
    end;
end.
```

```
                public shell
shell_data      segment public
local_word      dw  (?)
local_byte      db  (?)

shell_data      ends

all_code        group shell_code

shell_code      segment public
                assume cs:all_code,  ds:shell_data
shell           proc near

                nop
                nop
                nop



               "put code here"



                nop
                nop
                nop

shell           endp
shell_code      ends
                end
```

Continued...

C & PASCAL PROGRAMMERS

Blaise Computing provides a broad range of programming tools for Pascal and C programmers, with libraries designed for serious software development. You get carefully crafted code that can be easily modified to grow with your changing needs. Our packages are shipped complete with comprehensive manuals, sample programs and source code.

C TOOLS PLUS
$175.00

NEW! Full spectrum of general-purpose utility functions; windows that can be stacked, removed, and accept user input; interrupt service routines for resident applications; screen handling including EGA 43-line text mode support and direct screen access; string functions; and DOS file handling.

PASCAL TOOLS/TOOLS 2
$175.00

Expanded string and screen handling; graphics routines; easy creation of program interfaces; memory management; general program control; and DOS file support.

VIEW MANAGER
$275.00

Complete screen management; paint data entry screens; screens can be managed by your application program; block mode data entry or field-by-field control. Specify C or IBM/MS-Pascal.

ASYNCH MANAGER
$175.00

Full featured asynchronous communications library providing interrupt driven support for the COM ports; I/O buffers up to 64K; XON/XOFF protocol; baud rates up to 9600; modem control and XMODEM file transfer. Specify C or IBM/MS-Pascal.

Turbo POWER TOOLS PLUS
$99.95

NEW! Expanded string support; extended screen and window management including EGA support; pop-up menus; memory management; execute any program from within Turbo Pascal; interrupt service routine support allowing you to write memory resident programs; schedulable intervention code.

Turbo ASYNCH PLUS
$99.95

Complete asynchronous communications library providing interrupt driven support for the COM ports; I/O buffers up to 64K; XON/XOFF protocol; and baud rates up to 9600.

RUNOFF
$49.95

NEW! Text formatter written especially for programmers; flexible printer control; user-defined variables; index generation; and general macro facility. Crafted in Turbo Pascal.

EXEC
$95.00

Program chaining executive. Chain one program from another even if the programs are in different languages. Shared data areas can be specified.

ORDER TOLL-FREE 800-227-8087!

BLAISE COMPUTING INC.

2560 Ninth Street, Suite 316 Berkeley, CA 94710 (415) 540-5441

Circle Number 2 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

PORTABLE POWER WITH MasterFORTH

Whether you program on the **Macintosh**, the **IBM PC**, an **Apple II** series, a **CP/M** system, or the **Commodore 64**, your program will run unchanged on all the rest.

If you write for yourself, MasterFORTH will protect your investment. If you write for others, it will expand your marketplace.

Forth is interactive – you have immediate feedback as you program, every step of the way. Forth is fast, too, and you can use its built-in assembler to make it even faster. MasterFORTH's relocatable utilities and headerless code let you pack a lot more program into your memory. The resident debugger lets you decompile, breakpoint and trace your way through most programming problems. A string package, file interface and full screen editor are all standard features. And the optional target compiler lets you optimize your application for virtually any programming environment.

The package exactly matches *Mastering Forth* (Brady, 1984) and meets all provisions of the Forth-83 Standard.

MasterFORTH standard package	$125
(Commodore 64 with graphics)	$100
Extensions	
Floating Point	$60
Graphics (selected systems)	$60
Module relocator (with utility sources)	$60
TAGS (Target Applic. Generation System)– MasterFORTH, target compiler and relocator	$495
Publications & Application Models	
Printed source listings (each)	$35
Forth-83 International Standard	$15
Model Library, Volumes 1-3 (each)	$40

(213) 821-4340

MICROMOTION

8726 S. Sepulveda Bl., #A171
Los Angeles, CA 90045

Code for James M. Smith's "Turbo Pascal Inline Code"

```
program native;
{$K-}
{$C-}

var

s,t: text;
thing: array[1..256] of string[32];
thing2:array[1..256] of string[96];
i,j,k,limit: integer;

begin

    assign(t,'result.pas');
    assign(s,'shell.lst');
    reset(s);
    rewrite(t);

    writeln(t);
    writeln(t,'inline(');

    {this program loads MASM's .LST output, splitting each line
    {into a code and a comment half for processing.}

    for i:=1 to 22 do readln(s);    {scan off first 22 lines}
    limit:=1;
    while not eof(s) do             {read rest into arrays}
     begin
      readln(s,thing[limit],thing2[limit]);
      limit:=limit+1;
     end;                           {scan backwards to eliminate end}
    while (thing[limit][2]<>'0') and (limit>0) do limit:=limit-1;
    limit:=limit-6;
    while (thing[limit]='') and (limit>0) do limit:=limit-1;

    for i:=1 to limit do     {go through array line at a time}
    begin
     if thing[i][2]='0' then        {is line code?}
       begin
         delete(thing[i],1,7);      {remove address}
         j:=1;
         while j<=length(thing[i]) do   {remove non-code symbols}
          if not(thing[i][j] in ['0'..'9','A'..'F']) then
            delete(thing[i],j,1)
           else
            j:=j+1;
         if thing[i]<>'' then
         begin
           write(t,'$');    {send inline symbols}
           k:=3;
           while k<length(thing[i]) do   {insert further symbols}
             begin
               insert('/$',thing[i],k);
               k:=k+4;
             end;

           write(t,thing[i]);       {send code}

           if thing[i]<>'' then  {withhold slash if final line}
             if i<limit then
                   write(t,'/') else write(t,' ');
          end
            else write(t,'  ');
                                  {pad line with spaces}
          for k:=length(thing[i]) to 24 do write(t,' ');

          write(t,'{');   {start comment}
          j:=1;
          while j<=length(thing2[i]) do
           if thing2[i][j]=chr(9) then    {pad out tabs}
             begin
               delete(thing2[i],j,1);
               insert('        ',thing2[i],j);
             end
              else
               j:=j+1;
          write(t,thing2[i]);       {write comment}

              {does comment line wrap?}
          if (thing[i+1][2]='0') or (i=limit-1) then
            begin
              for j:=length(thing2[i]) to 48 do write(t,' ');
              writeln(t,'}');
            end;

        end   {if thing[i][2]='0'}
          else
            if thing[i][1]=chr(9) then  {wrapped comment}
              begin
                delete(thing[i],1,4);   {4 chr(9)'s}
                write(t,thing[i]);
                writeln(t,'}');
              end;

     end;  {for i:=1 to limit}

     writeln(t,'       );');
     writeln(t);
     writeln(t);
     close(t);
     close(s);

  end.
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Assembler Source Code for DOS Function Calls

by Stephen K. O'Brien and Wing Kit Cheung

The January/February 1986, issue of *Programmer's Journal* featured an extremely useful article ("Using DOS To Call Programs From Turbo Pascal") by Mr. Arvind Kumar. Mr. Kumar provided both the source code for and a lucid explanation of how to call other programs from within Turbo Pascal. Calling external programs from within a Turbo program is not only a neat trick, but an extremely useful tool for professional level programs.

I was disappointed, however, when the program didn't work. While I could start up another program from within the Turbo Pascal shell, I could not return to the shell. Some memory was being trashed, forcing me to reboot. At the end of his article, Mr. Kumar hinted at this problem: "If your program hangs consistently after the child program terminates, refer to Bela Lubkin's EXEC.PAS program in Data Library 1 of the Turbo Pascal Interest Group on CompuServe." Apparently, Mr. Lubkin has provided inline code that supplants the Turbo built-in MsDos procedure. I don't use CompuServe, however, and I don't find using inline code very instructive. Therefore, I am providing the Assembler source code that resolves the DOS Function call problem while demonstrating how to save the registers and call the DOS function. In fact, this procedure can be used for any DOS function call with greater safety than in using the Turbo Pascal MsDos function.

```
CODE SEGMENT
    ASSUME  CS:CODE
DOSFC4B    PROC    NEAR
       JMP     START

;*******************************************************
;
; This routine allows any command to be executed from
; within a TURBO Pascal program. The TURBO predefined
; procedure MSDOS cannot be used in this case because
; the DOS Function Call 4B(Hex) destroys most registers
; at return, including SS and SP.
;
; This routine is set up to be called from a TURBO Pascal
; program as a External sub-routine.
;
;*******************************************************

SSSAVE    DW      ?       ; SAVE SS
SPSAVE    DW      ?       ; SAVE SP
BPSAVE    DW      ?       ; SAVE BP

START:
       PUSH    BP          ; Save BP
       PUSH    DS          ; Save DS
       MOV     BP,SP
       MOV     SI,[BP+6]     ; Regs Offset
       MOV     ES,[BP+8]     ; Regs Segment
       MOV     BX,ES:[SI+2] ; Get values of BX, DX, DS and ES
       MOV     DX,ES:[SI+6]
       MOV     DS,ES:[SI+14]
       MOV     ES,ES:[SI+16]
       MOV     AX,SS       ; Save SS, SP and BP
       MOV     CS:SSSAVE,AX
       MOV     AX,SP
       MOV     CS:SPSAVE,AX
       MOV     CS:BPSAVE,BP

       MOV     AX,4B00H      ; Set for Function call 4Bh
       INT     21H

       MOV     SS,SSSAVE     ; Restore SS, SP and BP
       MOV     SP,SPSAVE
       MOV     BP,BPSAVE
       MOV     SI,[BP+6]     ; Regs Offset
       MOV     ES,[BP+8]     ; Regs Segment
       MOV     ES:[SI],AX    ; Set Return code
       PUSHF
       POP     AX
       MOV     ES:[SI+18],AX      ; Set Flags

       POP     DS
       POP     BP
       RET     4

DOSFC4B    ENDP
CODE          ENDS
       END
```

To use this procedure, you must declare the external procedure DOSFC4B as follows:

```
DOSFC4B( Var Regs : RegType); External 'DOSFC4B.COM';
```

Regtype is the familiar 8088 register set record used in numerous Turbo Pascal programs. It is defined in Mr. Kumar's procedures in this form:

```
Type
  RegsType = Record
             AX,BX,CX,DX,BP,SI,DI,DS,ES,Flags : Integer;
             end;
```

The calling format of the procedure is

```
DOSFC4B( Regs );
```

Below is a listing of the main body of the Function named Fork. The old code, using the function CallDOS, has been replaced with a call to the assembler routine DOSFC4B. □

```
BEGIN                       (* Function Fork *)
Status := True;
MakeFCB(CmdLine, FCB1, FCB2);
ProgName := ProgName + Null;
CmdLine  := ' ' + CmdLine + CR;
CmdLen   := Pred(CmdLen);

WITH ParmBlock DO
  BEGIN
  EnvSeg     := EnvSegAbs;
  CmdPtr     := Addr(CmdLine);
  FCB1Ptr    := Addr(FCB1);
  FCB2Ptr    := Addr(FCB2);
  END;

Regs.BX := Ofs(ParmBlock);
Regs.DX := Ofs(ProgName[1]);
Regs.DS := Seg(ProgName[1]);
Regs.ES := Seg(ParmBlock);

{----- O L D   C O D E -----}
{
if (NOT CallDOS(Regs)) then
  begin
  Status := False;
  WriteLn('Unable to run "', ProgName, '" with "',CmdLine, '" : ',
    DOSErrorMsg(Regs.AX));
  end;
}

{----- N E W   C O D E -----}
DOSFc4B(Regs);

IF (Odd(Regs.Flags)) THEN
  BEGIN
  Status := False;
  WriteLn('Unable to run "', ProgName, '" with "',CmdLine, '" : ',
    DOSErrorMsg(Regs.AX));
  END;

PgmCall := Status;
END;                        (* Function Fork *)
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Alignment Problems in C and Managing Memory Allocation

by William James Hunt

Alignment Problems in C

Most of the time a programmer can write and test C programs without much concern for the peculiarities of the underlying computer hardware. But sometimes a C programmer needs to understand a hardware limitation and its effect on his work. Aligning variables on even byte addresses is such a requirement.

The IBM PC's 8088 processor fetches and stores data 8 bits at a time, but faster microprocessors and larger computers access data in 16- or 32-bit chunks. In such a computer, 16-bit words must be located at even byte addresses (0 and 2, for example) to be fetched or stored in a single access. Older computers did not allow data values larger than 8 bits to be located at odd byte addresses (1 or 3, for example). Newer computers allow words to be located at odd byte addresses—they perform two memory accesses per word. Either way the computer's limitation affects the C compiler and may cause bugs in your programs.

C compilers must generate programs that fit the alignment restrictions of the computer on which they will be executed. On the Intel 8086 processor family, alignment is not mandatory, but programs will run faster on 8086 and 80286 processors if 16-bit data is word aligned. The Microsoft C 3.00 compiler always forces word alignment, and Lattice C offers word alignment as an option. The default for Lattice C 3.00 is byte alignment (16-bit data is not forced to be word aligned). This may produce slightly smaller programs, and it is necessary when structures defined by other programs or PC DOS with unaligned members must be handled.

Aligning single variables on even byte address causes no problems, but aligning 16-bit members within a structure can cause bugs. The s1 structure below shows the problem that can occur:

```
struct          /* Unsafe Structure */
 { char c1 ;
   int  i1 ;
 } s1 ;
```

When this structure is compiled with byte alignment specified, the integer member i1 is located one byte from the start of s1. But when word alignment is specified, i1 is located two bytes from the start of s1. As long as all source files that use s1 are compiled with the same alignment, no problems will appear. But this is a time-bomb waiting to go off. If a single source file is compiled with a different alignment specified, the program may develop a bug. The bug may also be triggered by switching to a new version of the compiler (or a different compiler) that uses a different default alignment.

Unsafe structures may also cause problems in using compiler library functions and add-on libraries. When you have problems with library functions, check the associated header files (*.H) to see if your program communicates with library functions with unsafe structures. Check the library documentation or call the supplier for the alignment option that must be used with such structures.

You can define safe structures that avoid alignment time-bombs. Arrange structure members so that any non-char members are explicitly located an even number of bytes from the beginning of the structure. This ensures that all the structure's members will be located at the same offsets whether byte or word alignment is specified. You may have to define one or more dummy char members to force non-char members to even byte offsets. Grouping all char members together reduces the number of dummy members needed. In the s1a structure below, the dummy member dum1a ensures that i1a is located two bytes from the start of s1a.

```
struct          /* Safe Structure */
 { char c1a ;
   char dum1a ;
   int i1a ;
 } s1a ;
```

Some older computers with a 32-bit data path required 32-bit data values to be located at addresses divisible by 4 (0,4,8, for example). You may not need to force 32-bit values to 4 byte alignment, but you should remember that the potential problem exists.

Some simple rules can avoid bugs caused by alignment problems:

1. When you define structures, make them safe by forcing word alignment on 16-bit or larger members.
2. When you must use unsafe structures defined by other programs or PC DOS, confine their use to one or a few source files.
3. Document your overall assumptions about alignment. For example, if you force 32-bit members to even byte offsets rather than 4 byte offsets, document your assumption. Document your choice of compiler alignment option. If you are using a MAKE program, specify the alignment choice on the command line. List any files that contain unsafe structures and the alignment they require.
4. Don't use numbers for the size of a structure or the offset of its members. Use the sizeof operator when the structure's size is needed, and refer to members by name (p->i1) rather than with a numeric offset (*(p+1)).

Managing Memory Allocation

The Lattice and Microsoft C libraries provide useful memory allocation functions, but there are good reasons for confining their use to a single interface module:

- As you refine a program, you may need to change the memory allocation functions you use. This is feasible if the change is confined to a single source file.
- Lower level memory allocation functions may be useful, but

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

they are not standard, so you don't want to embed references to them throughout your programs.

- Bugs in your use of library memory management functions can be hard to find.

The memfun.h and memfun1.c source files below isolate your programs from the library functions and provide debugging help. The tot_all variable keeps track of the amount of memory currently allocated. Since it is a global variable, it can be printed from other functions or examined from a symbolic debugger such as C-Sprite or Symdeb. The malloc and free functions are used here, but this can be changed without affecting the rest of your program.

When DEBUG is defined on the command line with the -d DEBUG switch, the compiled memall and memfree functions provide debugging output each time they are called. Pointers are printed as integer or long length values, depending on the memory model used. The memall function can be forced to return a NULL value, providing a convenient test of error handling in the calling function. If the symbol DEBUG is undefined when memfun.h is compiled, the debugging statements are not compiled. □

William Hunt writes the Lattice C Users' Group *newsletter, where this material first appeared, and is also the author of* The C Toolbox, *published by Addison-Wesley. Readers may contact Bill at the following address:*

William James Hunt
Lattice C Users' Group
P.O. Box 271965
Concord, CA 94527

Code from William Hunt's "Managing Memory Allocation"

```
/* memfun.h - memory allocation interface */
#ifndef MEMFUN                  /* once only test */
#define MEMFUN 1

#ifndef MYANSI
#include "myansi.h"
#endif

#ifdef FPRO                     /* prototypes */
char *memall(unsigned) ;
void memfree(char *,unsigned) ;
#else                           /* return values only */
char *memall() ;
void memfree() ;
#endif

#endif                          /* end of once only test */

/* memfun1.c - our dynamic memory allocation interface */
#include "stdio.h"
#include "myansi.h"
char *malloc() ;
void free() ;
long tot_all  = 0L ;            /* no. bytes currently allocated */

char *memall(n)                 /* allocate a memory area */
 unsigned n ;                   /* size of area in bytes */
 {
   char *p ;
   char s[81] ;

#ifdef DEBUG
   printf("\n memall called - n=%u Dec %x Hex\n",n,n) ;
   printf(" return NULL (y/n) \n") ;
   fgets(s,80,stdin) ;          /* force allocation failure ? */
   if( (s[0] == 'y') || (s[0] == 'Y') )
        return( NULL ) ;
#endif

   p = malloc(n) ;              /* get some memory */
   if( p != NULL ) tot_all = tot_all + n ;

#ifdef DEBUG
   if( sizeof(char *) == sizeof(int) ) /* display memory address */
        printf("  p=%4x " ,p);
   else printf("  p=%41x ",p);
   printf("\n %81d Bytes Allocated \n",tot_all) ;
#endif

   return( p ) ;
 }

void memfree(p,n)               /* free allocated area */
 char *p ;                      /* address of area to free */
 unsigned n ;                   /* size of area in bytes */
 {
   free(p) ;                    /* free the memory */
   tot_all = tot_all - n ;

#ifdef DEBUG
   printf("\n memfree called - n=%u Dec %x Hex",n,n) ;
   if( sizeof(char *) == sizeof(int) ) /* display memory address */
        printf("  p=%4x " ,p);
   else printf("  p=%41x ",p);
   printf("  %81d Bytes Allocated \n",tot_all) ;
#endif
 }
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

DESMET C EXTRAS

STRAIGHT FROM OUR CAST OF THOUSANDS

WELL...AT LEAST TEN OR SO.

BBS-1: Programs from our BBS incl. ARC, WINDOW, EXE2COM.C, DOSEDIT, EXE2ROM & more **$8**

BBS-2: ARC, O88 (C88 optimizer), CALENDAR, & more **$8**

D88: source level debugger **$50**

DOS LINK OPT: converts .O format files to .OBJ format **$35**

GRAPHICS: high/med/low res graphics (with source) **$35**

HACKER: source for _CSETUP, DOS 2.0 I/O, RAM.COM, a program to dump .O files & a little more **$25**

SEE[tm] : full screen editor (provided free with compiler) **$50**

MAKE: full UNIX-level MAKE. Recreates any system from its specified sources. Full macro & path name support. This product is not DeSmet specific **$50**

TOOLS: Tools and Utilities that DeSmet and Ouye use. (with Source) **$35**

XARRAY: a DOS eXtended ARRAY pkg which lets you define and access data arrays in the RAM above your C address space. Arrays any size to avail. memory. (With Source) **$39**

DESMET C FOR THE MACINTOSH: full K&R C w/ mach. level debugger. **$150**

DESMET C FOR THE PC — compiler, binder, assembler, std. and 8087 libs., SEE[tm] (full screen editor), Clist, Librarian, Compare and Profile **$109**

DESMET C- LARGE CASE — compiler (PC) as above plus D88 (debugger) and Large Memory Option **$209**

C WARE
CORPORATION
P.O. Box C
SUNNYVALE, CA 94087 USA
(408) 720-9696 TELEX: 358185
Street Address: 505 W. Olive, #767
VISA, MC & AMEX

Circle Number 38 on the Reader Service Card

MicroSoft provides the CodeView - Atron provides the ProbeView

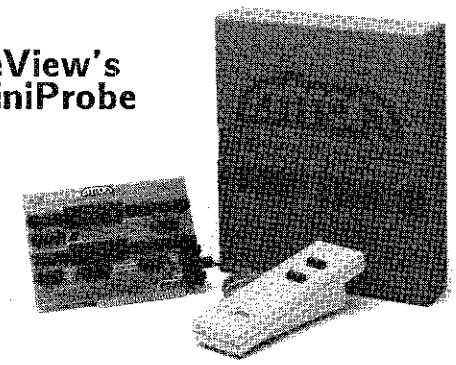
Now you can get hardware assisted software debugging support from Atron for the CodeView debugger you may already have. Atron, maker of the debugging tools used by 9 out of the top 10 software developers in the PC market has expanded its debugging support to let you use Microsoft's new Codeview software debugger with all of Atron's hardware PROBES.

pointers which overwrite the program code or data. This problem is particularly insidious in that it normally overwrites in a different place after each new compile of your program.

MINIPROBE'S hardware breakpoint can also be on a range of memory locations. This helps trap events like uninitialized pointers.

with CodeView's debut -MiniProbe too!

$395



Now Atron PROBES have a little brother, MINIPROBE. MINIPROBE inherits its hardware assisted software debugging features from Atron's other PROBES the AT PROBE and PC PROBE, but is lower in cost. What can MINIPROBE do:

MINIPROBE can trap events like reading and writing to memory or IO devices in real time with its hardware assisted breakpoint. This solves the most common problem in software development today - out of range

When the computer is locked up, you're locked out from debugging. MINIPROBE has a crash recovery switch box that lets you regain control to see what went wrong.

So why waste more time slaving all day over a nasty software bug get an Atron debugging PROBE today!. (408)-741-5900

Atron
THE DEBUGGER COMPANY
20665 Fourth Street • Saratoga, CA 95070

Trademarks: CodeView - MicroSoft, MiniProbe,AT Probe - Atron

Circle Number 1 on the Reader Service Card

1	2
3	4

Have we got your number?

Is Programmer's Journal reaching you at the correct address? Please notify us of any address change or correction.

Is it nearing time to renew? Your Programmer's Journal mailing label shows when your subscription expires. Renew early to avoid any interruption in your subscription service!

Programmer's Journal, P.O. Box 3000, Dept. EE, Denville, NJ 07834

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

ED/ASM-86

Now with Full Screen Editor Built in!

The first truly integrated Editor, Assembler, Debugger, and Linker is now available from Oliver Computing Company.

*** The Editor supports block move, block copy, partial save and partial load insertion. All text is tokenized for both efficiency of storage and very fast assembly time. FAST screen I/O and separate editor/debug windows.**

*** The Assembler can be immediately invoked after any editting command. Output can be directed into memory, to a .COM file, .EXE, or to ED/ASM-86's own efficient .LNK file format. 8087/186/286 instructions supported. Macros and most standard pseudo-ops are supported.**
UNIQUE to ED/ASM is its structured assembly language capability. No more wasted labels and time! Multiple segmented programs can be be sent to .COM files.

*** The built in debugger supports symbolic disassembly, integration with the assembler, as well as disassembly of 8087/186/286 instructions. Display 8087 registers in scientific decimal format. SPLIT SCREEN debugging with a dynamic register window is easily invoked. Disassemble programs to disk file for easier analysis. "Single-step" data editting supported for bytes, words, double words, and 8087 floating constants in memory. You can "patch" assemble a line at a time with symbols from previous assembly (even define symbols as you go!)**

*** The linker supports ED/ASM-86 link files, which can be created from virtually any valid ED/ASM-86 source file, even lines with multiple external symbols in expressions. (Note: not compatible with usual link format)**

*** And many more features too extensive to be listed here.**

All of these features exist in one integrated program. A typical development cycle with ED/ASM-86 consists of entering a program with the editor, or editting an existing program, typing "A:,-IM" to assemble into memory, going into the debugger to immediately test the program. Then you go back to the editor for the next cycle.
ED/ASM was developed for Oliver Computing's own use, and has been used for over a year to enhance itself!

ED/ASM-86 is only $95. Send check or M.O. to:
Oliver Computing Company, P.O. Box 90140, Indianapolis, IN 46290 { (317) 849-4450 }
for immediate delivery via UPS.

Circle Number 15 on the Reader Service Card

Program faster, debug faster, with Source Print™

SOURCE PRINT makes your job easier... gives you clear, organized source code!

- Draws lines around nested structures
- Automatic indentation
- Index (cross-reference)
- Table of contents which lists functions, procedures & subroutines
- Easy-to-use menu
- Find files that contain a given string
- Prints keywords in boldface
- Splits multi-statement BASIC lines into 1 statement per line.
- Handles up to 50 input files & over 60,000 lines of code.
- For dBASE, C, BASIC, Pascal and Modula-2.

"Here is possibly the ultimate source code formatting utility" —Data Based Advisor, July '86

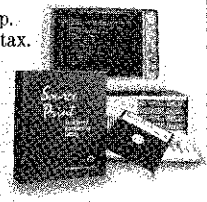
"No one can argue the importance of having well-documented, legible source code. And Source Print delivers just that."
—PC Magazine, Sept. 16, '86

For IBM PC, XT, AT, compatibles.
Not copy protected.
Immediate delivery.
MC, Visa, AmEx, COD

$139.00 +$5 ship.
CA residents add tax.

Call to order
800-257-5773
(outside CA)
800-257-5774
(within CA)
or see your local dealer

Source Print
Ver. 2.0
Comprehensive Formatting Tool
from Aldebaran Labs
3339 Vincent Rd., Pleasant Hill, CA 94523



Are your programs less than 1000 lines? This is for you:

NEW!
COMPACT SOURCE PRINT

COMPACT SOURCE PRINT handles up to 1000 lines of code & up to 10 files. You get all SOURCE PRINT features except bold keywords and table of contents.

only **$55.00** Introductory Price

Circle Number 52 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

The Proposed ANSI C Language Standard, Part 4

by Rex Jaeschke

© Rex Jaeschke, 1986. All rights reserved.

The Chicago ANSI Meeting

The most recent ANSI C Committee meeting was held from September 8–12 in St. Charles, IL. Our meeting host was Lattice, Inc., represented by VP Steve Hersee. Steve is also the convener of the recently formed ISO C Language Standards Committee and that group's liaison with ANSI X3J11.

The most significant outcome of the meeting is that the Committee unanimously voted to forward the Draft Standard to X3 for public comment. Comments will then be accepted for four months with each receiving a formal, written response. After that review period expires, all necessary changes will be made and the Draft will be re-released for a second review period (of two months only), after which it will be voted on by the full X3J11 Committee (and the X3 Secretariat) before becoming an official ANSI Standard.

This whole process may be completed by the third quarter of 1987, but as much of the current Draft Standard is unlikely to change, MS DOS (and other) compiler implementers are already beginning to provide many of the new capabilities. Of course, until there is an official Standard there can be no conforming implementations. So compilers may have to "undo" some of their new features if these are removed from the final Standard, or make them extensions, provided they don't conflict with the Standard and its future directions.

Copies of the Draft Standard and the accompanying Rationale Document are available from

X3 Secretariat CBEMA
311 First Street, N.W.
Suite 500
Washington, D.C. 20001
(202) 737-8888
Cost: approx $40

ftell and fseek Limitations

IBM (and other) representatives indicated the limitations of using ftell and fseek in mainframe environments where files are record-oriented (as opposed to byte-streams) and file sizes may range up into the tens of gigabytes.

The problems with these functions are as follows: For historical reasons, ftell returns the current file position as a long int, which for most machines is a signed 32-bit value. This effectively places an implementation-defined limit that is insufficient for that implementation's file system. Also, the byte-stream model is not well-suited to record-oriented file systems such as those used by IBM mainframe operating systems and DEC's PDP-11 and VAX/VMS environments.

To remove this file size restriction and to make the Standard upwards compatible, the Committee added two new functions to the Standard Library. fgetpos returns the current file position for a given stream. The return type of fgetpos is a derived type called fpos_t (which need not be an integral type). The function fsetpos sets the file position indicator to the value provided, such value (of type fpos_t) having first been obtained by a previous call to fgetpos.

Internationalization Issues

The majority of the show-stopping votes in the previous attempt to start the public comment process grew out of the inadequate ability to implement and/or use a conforming compiler in non-English environments.

Since ANSI is The American National Standards Institute, why is an ANSI Standard bothering with countries outside the U.S.? On the surface, that seems a reasonable question, but the Committee had good reasons for broadening their horizon.

Firstly, a number of X3J11 voting members represent other countries: Canada and the U.K. immediately come to mind, and numerous European companies are at least observing members. One doesn't have to be a U.S. citizen, resident, or company to be an ANSI voting member. One must simply maintain a given meeting attendance record and pay the necessary fee.

Secondly, many of the voting members sell, or plan to sell, their products to non-English speaking countries. Certainly, Microsoft and Lattice have a vested interest in the Asian market, while very large companies such as IBM, DEC, Sperry, and AT&T are already heavily committed to the European marketplace.

Thirdly, with the advent of an International C Language Standard, via ISO, it is within ANSI's voting members' best interests that the ISO Standard be a proper superset of the ANSI Standard so that they don't need different versions of the compiler and/or runtime library for different countries.

For these (and possibly other) reasons, the ANSI Committee felt compelled to at least provide the hooks in their Standard so that users could write conforming programs and at the same time resolve their internationalization problems.

It is fairly easy to demonstrate that all of the issues cannot be solved by algorithms; there are just too many exceptions and not enough rules. The main problems are as follows:

1. The national character set: What happens to isupper and islower if a character does not have both cases? Then there are the European characters with umlauts, accents, etc., and the middle-eastern and Asian (multi-byte) characters.

2. Collating sequences: There are enough different ways of sorting things in an English environment without even looking at foreign language issues. Methods include dictionary sort, phone directory sort, letters before or after numbers, and positions of various punctuation groups (ASCII versus

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

EBCDIC).

3. Numeric editing: The methods of representing positive and negative integral and floating-point values vary widely, particularly with regard to position of sign (if any) and the thousands and decimal point separator characters. Rounding is also handled in various ways.

4. Currency symbols and position: Many countries use different methods for currency and non-currency numeric values.

5. Date and time formats: You name it—someone uses it and a whole lot more formats you never dreamed of.

6. Some countries don't use just one international environment: Switzerland, for example, has about five different languages, depending on your location, and various combinations of date and time and numeric formatting. The rules can become very complicated or impossible.

In order to preserve the existing, large body of code, it was decided to place the burden of having a non-USA/ENGLISH environment only on those users who needed it. This will be done by making the C program startup environment default to a native "locale" of USA. Users wanting something else will need to call the setlocale function to set their locale to some other implementation-defined environment. The intent is to let a program switch from one locale to another at runtime as necessary.

Future Meetings

The schedule for future ANSI meetings is

Dec '86 Atlantic City, NJ (Plum Hall)
Mar '87 Boulder, CO (AT&T)
Jun '87 Paris, France (COSMIC)

Trigraphs Revisited

Last issue (Vol. 4, No. 5), I discussed the addition of trigraphs. To demonstrate trigraphs further, I have included the following program. Its meaning is left as a reader exercise.

```
/* a valid program using trigraphs */

??=include <stdio.h>

??=define TRUE 1
??=define FALSE 0

int array??(4??)??(2??) = ??<
        ??<1, 2??>, ??<3, 4??>,
        ??<5, 6??>, ??<7, 8??>
??>;

struct stuff ??<
        char text??(10??);
        int count;
??>;

f()
??<
        FILE *infile;
        int j = 7, k = 077;

        infile = fopen("??/??/file.dat","r");

        if (j == 0 ??!??! j ??! 1)
                j ??!= k;
??>
```

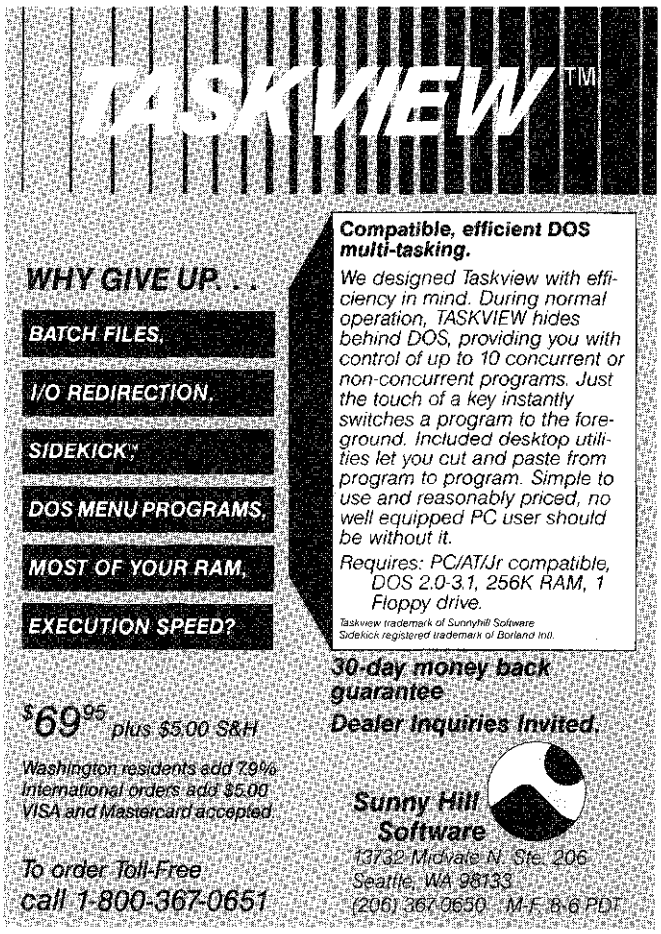
From this example, it should be fairly obvious that trigraphs should be used only when *absolutely* necessary. □

Rex Jaeschke is an independent computer consultant, writer, and lecturer. He is also a member of the X3J11 Committee and co-founder and Editor of the C Journal.

If you would like to comment on or question any aspect of the Proposed ANSI C Standard, write to

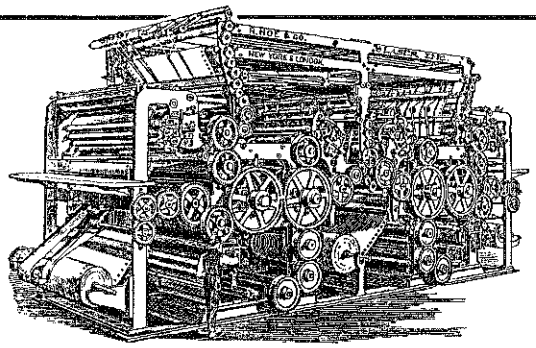
Rex Jaeschke
2051 Swans Neck Way
Reston, VA 22091

Rex will endeavor to address your concerns in future columns as space permits.


TASKVIEW™
WHY GIVE UP . .
BATCH FILES,
I/O REDIRECTION,
SIDEKICK,*
DOS MENU PROGRAMS,
MOST OF YOUR RAM,
EXECUTION SPEED?
Compatible, efficient DOS multi-tasking.
We designed Taskview with efficiency in mind. During normal operation, TASKVIEW hides behind DOS, providing you with control of up to 10 concurrent or non-concurrent programs. Just the touch of a key instantly switches a program to the foreground. Included desktop utilities let you cut and paste from program to program. Simple to use and reasonably priced, no well equipped PC user should be without it.
Requires: PC/AT/Jr compatible, DOS 2.0-3.1, 256K RAM, 1 Floppy drive.
Taskview trademark of Sunnyhill Software
Sidekick registered trademark of Borland Intl.
$69.95 plus $5.00 S&H
Washington residents add 7.9%
International orders add $5.00
VISA and Mastercard accepted
To order Toll-Free
call 1-800-367-0651
30-day money back guarantee
Dealer Inquiries Invited.
Sunny Hill Software
13732 Midvale N. Ste. 206
Seattle, WA 98133
(206) 367-0650 M-F, 8-6 PDT

Circle **84** on the reader service card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



The Canon A1 Laser Printer

by Paul Barkley

Coordinating the Canon A1, HP font cartridges, Word Perfect, and pitch, point size, proportional spacing. . . .

Laser printers are remarkable hybrids that share similarities with dot matrix printers, letter quality printers, plotters, typesetting machines, and of course copiers—all rolled into one. When you buy a laser printer, you get all the advantages and the disadvantages of this equipment assortment. Advantages include the quality and proportional spacing ability of a letter quality printer; the copier equivalent of a cut sheet feeder; the ability of a dot matrix printer to easily switch type faces; the speed of a copier (8 pages per minute and much higher); and the ability to draw lines and shapes almost like a plotter. Disadvantages include an inability to handle multiple part forms or forms of irregular size such as envelopes; an inability to handle regular gummed label stock in one-up or two-up configurations; and the fact that you can't print part of a page and see what has been printed without ejecting it.

Laser printers are available in an astounding range of prices and features, from about $2,000 to several hundred thousand dollars, and although the quality, in terms of dot resolution, is surprisingly about the same across the entire spectrum, the more expensive models have much higher through-puts and are capable of much more complex data manipulation.

For about $2,500 you can buy a Hewlett-Packard LaserJet, a Canon A1, or one of a number of other inexpensive laser printers. These are entry level machines and offer a quantum improvement (in my opinion) over printing options such as dot matrix printers.

For about $3,500 you can move up to models that offer more memory so that larger and more complex graphics can be printed. The HP LaserJet Plus is a good example. But if you don't plan on printing a lot of graphics material, the extra capacity may be wasted.

For about $5,000 you can buy a Xerox 4045, which uses regular toner instead of the handy but expensive toner cartridges used in most smaller laser printers. The 4045 has an option that lets it function as a copier. Single sheets can be fed into a mechanism and copied, and although the copies are of excellent quality, you can't copy anything as bulky as a book.

Since many of the smaller lasers use the Canon printing engine, their quality tends to be uniformly good. You'll be very pleased with clarity and resolution, but if you're not planning on spending $5,000 on a laser printer, do yourself a favor and don't look at the Xerox 4045. The Canons and HPs look fabulous by themselves, but they come off badly in a side by side comparison with Xerox. The 4045 simply lays down a blacker ink that is absolutely stunning. Period.

If you want to spend $7,000 or $8,000, you can move up to Apple's LaserWriter. This machine boasts quite a few improvements over smaller models, including much more intelligent font manipulation.

My excuse for buying a laser printer was to upgrade the quality of the documentation I supply to clients with their custom software. Typically, these custom systems need about 80 pages of user documentation. This is enough material to require division into major sections and subsections, and it was difficult to establish visual separation using only a near letter quality printer. What I needed was a way to use a variety of *sizes* of type, from small print for footnotes to large headline type.

In addition, since almost all of my packages are written on a fixed fee basis, it's crucial to write the documentation and get it printed in as short a period as possible. It used to take several hours to run a final copy on continuous form paper, burst it, hole punch it, and mount it in a notebook for the client. Not only do laser printers print higher quality at a greater speed than dot matrix printers, but they also use cut paper, which doesn't have to be burst. For a little more cost, you can use pre-punched paper and save yourself the aggravation of hole punching as well.

Note that this ability to use paper the way a copier does also has its drawbacks. If you want to print labels, you'll have to buy special copier label stock—11 rows of 3-up labels. Unfortunately, you can't print all of the area in the top and the bottom rows, so you'll waste 6 out of every 33 labels.

Envelopes are also a problem—they're too wide to fit in the printer. I solved this problem—this sounds much worse than it is—by extracting the name and address from a letter and printing it on a separate sheet of paper. The procedures for this in Word Perfect are relatively automatic. Then I take the sheet over to my Canon PC-24 copier, throw an envelope in the feed, and copy the address onto the envelope. The type on the envelope then matches the type in the letter. The alternative is to type envelopes by hand or use labels of some sort. This is one of the trade-offs.

When I began my search for a laser printer, I knew exactly what I wanted to use it for. Graphics were relatively unimportant to me, but a selection of proportional fonts in at least three sizes was crucial.

After looking through all the font catalogs, I found exactly what I wanted: Hewlett-Packard's font cartridge "B." B combines a proportional Times Roman typeface in both 8 and 10 point with a Helvetica Bold for headlines in 14.4 point. ("Points," which are height measurements of 1/72nd of an inch, are discussed below. Learning printer's measures is like learning hexadecimal arithmetic—without an underlying number scheme. It makes you seriously consider abandoning everything for the metric system.)

Having selected the fonts, I tearfully bid adieu to good old WordStar in order to move on to Word Perfect, which is far more agile and handles fancy font changes easily. All that re-

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

mained was to locate a printer that could use the cartridge and which would run under Word Perfect.

Although the Hewlett-Packard LaserJet seemed like the logical choice, I found a more attractive deal on a Canon A1 from a VAR who knew quite a bit about the machine. They assured me that the Canon would accept the HP cartridge although they were a little vague about how it could be done in terms of software—physically, the two machines are almost identical and the cartridges are interchangeable.

Apparently neither Canon nor HP approves of this mix and match approach, and I assumed it was for short sighted marketing reasons. But after I tried to use this combination with Word Perfect, I realized that an excellent reason for discouraging experimentation is to avoid endless discussions with end users who can't get their installations correctly configured. Therein lies a tale.

The Canon A1 Printer

When I think of the Canon manual, one word comes to mind: "inscrutable." Although the manual contains a great deal of information, it is difficult to put into perspective. Just as a dictionary used by itself is not sufficient to master a new language, the Canon manual is inadequate for mastering the many features of the printer. What's missing is an explanation of the over-all concept of accessing and manipulating fonts. When I finally figured out how it was done, I was aghast.

The secret to calling out a font is not in locating some unpublished, esoteric name or number that corresponds to the font, which is common in many other kinds of printers. Instead, fonts are called out the way you play hand grenades—just get close enough and you win. You get "close enough" by specifying a series of parameters that more nearly match the font you want than any of the fonts you don't want. A font, in this usage, is a specific typeface in a certain size, with a number of other parameters specified.

Before we talk about parameters, let me say that the Canon A1 *does* have a series of buttons in the front that will let you select your font manually. This assumes that you're capable of depressing more than one switch at the same time, at a pause rate of three seconds, while decoding meaningless symbols that appear on the two-character LED.

This physical selection process, however, works only for fonts that the printer is currently aware of. When you power up the printer, it's aware of only its built-in fonts. Any cartridge present is unknown. Thus, you must find a way to "teach" the printer about the cartridge fonts before you can physically access them. If you're going to do that, you might as well just select the font by software control.

Font Parameters

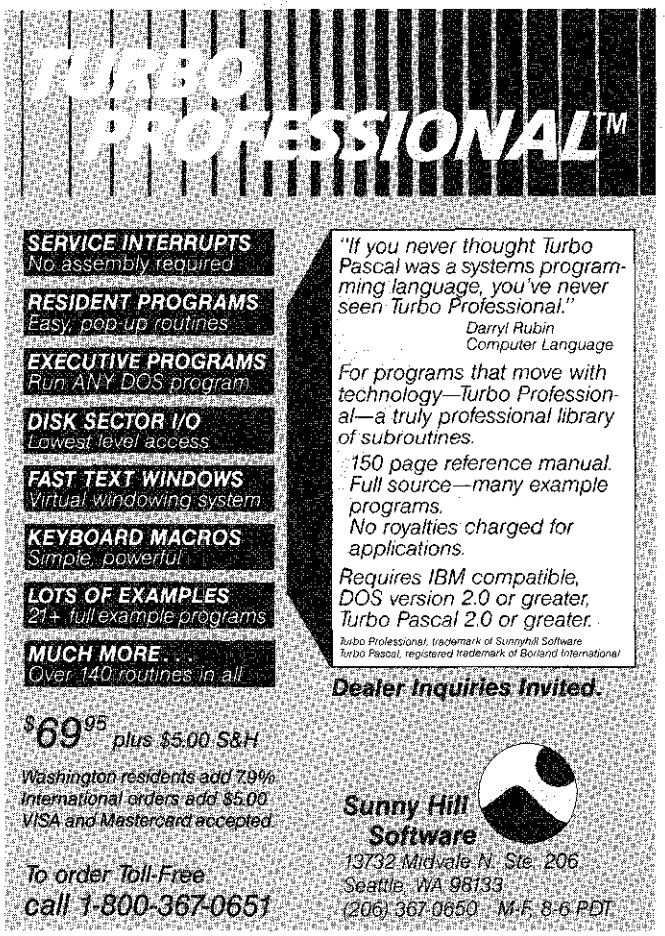
Font parameters include:

- Orientation
- Character Set
- Typeface
- Pitch
- Point Size
- Style
- Stroke Weight

These parameters are arranged in hierarchical order, with orientation having a greater effect than stroke weight. Thus, in our game of hand grenades, you can score a hit by getting most of the parameters at the top of the list correct and leaving a few off the bottom of the list.

"Orientation" refers to which way the characters are printed on the page. "Portrait" is the normal rotation; "landscape" can be read when the paper is turned on its side (wider than it's long). While some laser printers are smart enough to rotate a given typeface to print in either mode, on the Canon you must have separate typefaces for the different orientations. One small typeface that comes with the printer is in landscape orientation, and you can use this for your wide spreadsheets. Most typefaces are in the normal portrait orientation. Because of the copier characteristics of the printer (paper is fed from a cassette tray), you can't simply put paper in the machine sideways the way you can with a wide carriage impact printer.

"Character set" refers to a system of characters, or alphabet, peculiar to a nationality. Foreign character sets are a serious problem with many printers unless appropriate characters are available on daisy wheels or dot matrix ROMs. So the Canon A1 wants to know what character set you want, such as French or Spanish. I wanted a set called "USASCII" (a run-together of "U.S. ASCII"). This carries the secret code 66, which can be found, with great effort, in the manual.


Turbo PROFESSIONAL™
SERVICE INTERRUPTS
No assembly required
RESIDENT PROGRAMS
Easy, pop-up routines
EXECUTIVE PROGRAMS
Run ANY DOS program
DISK SECTOR I/O
Lowest level access
FAST TEXT WINDOWS
Virtual windowing system
KEYBOARD MACROS
Simple, powerful
LOTS OF EXAMPLES
21+ full example programs
MUCH MORE...
Over 140 routines in all
"If you never thought Turbo Pascal was a systems programming language, you've never seen Turbo Professional."
Darryl Rubin
Computer Language
For programs that move with technology—Turbo Professional—a truly professional library of subroutines.
150 page reference manual.
Full source—many example programs.
No royalties charged for applications.
Requires IBM compatible, DOS version 2.0 or greater, Turbo Pascal 2.0 or greater.
Turbo Professional, trademark of Sunnyhill Software
Turbo Pascal, registered trademark of Borland International
Dealer Inquiries Invited.
$69.95 plus $5.00 S&H
Washington residents add 7.9%
International orders add $5.00
VISA and Mastercard accepted
To order Toll-Free
call 1-800-367-0651
Sunny Hill Software
13732 Midvale N. Ste. 206
Seattle, WA 98133
(206) 367-0650 M-F, 8-6 PDT

Circle Number **85** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

"Typeface" refers to a generally copyrighted expression of the characters of the alphabet. Thus, when someone says he or she wants a brochure set in Helvetica medium, a typesetter knows exactly what is wanted. A character set in Helvetica does not look the same as one set in Times Roman, for example, and while most people can see that one typeface is different from another (and react to the appropriateness of a selection), a trained individual can also identify which typeface it is. (If you want to be a hit at contemporary cocktail parties, I suggest you learn to talk knowledgeably about typefaces. Just don't party with any printers or graphic artists.)

The Canon printer recognizes assigned numbers for typefaces, too, but the manual gives sparse indication of what these numbers might be. The Canon's built-in typeface is Courier, which is number 3, but one of Canon's assigned numbers, 10, not only works for their proportional (see below) cartridge font called Garland, but it also works for HP's Times Roman. By the way, HP calls this TMS RMN, presumably to avoid copyright conflict. Similarly, Helvetica is called HELV.

"Pitch" refers to the characters per inch (CPI). Most people comfortably recognize the difference on typewriters between 10 pitch (pica) and 12 pitch (elite). Most dot matrix printers support compressed pitches of about 16 characters per inch as well. When you lay out a document, you need to know how many characters per inch you'll be using so that you can see where columns will fall and where the margins should be.

For years, all we had on typewriters and printers was fixed pitch typefaces. If you type ten i's using a 10 pitch daisy wheel, they'll take up 1 inch of horizontal space. If you type ten m's using the same wheel, they'll take up the same amount of space. If you then examine the type, you'll notice that the i's have quite a bit of white space between them while the m's are all smashed together and hard to read. An m completely fills the space, while an i leaves room on either side. This is, well—unnatural.

Proportional typefaces, on the other hand, allocate exactly the right amount of space to each letter. Thus, *i*'s take up a small amount of space, while m's get more room to breathe. The result is much more readable type. In addition, you usually get more characters per page with a proportional typeface, assuming that what you write uses a normal distribution of letters.

If you think about it, though, you'll quickly realize that proportional type will sorely vex word processing software that is trying to right justify text (make the right margins even, like the left margins). You can handle this by using a smaller scheme of spacing. For example, an i might take up 16 units of measurement while an m takes up 38 units. These units are actually 1/300th of an inch in the Word Perfect scheme. In addition, this fine spacing also allows the clever word processing software to insert a little extra space between letters in right justification instead of throwing "gobs" of space between words as is the case with fixed pitch typefaces.

Note that different typefaces will have proportionally different letters. In one face, for example, i's and m's might be assigned widths of 18 and 37 instead of 16 and 38. In some typefaces, the i may be proportionally "fatter." Serifs—those little lines at the top and bottom of letter strokes in some typefaces—can make a big difference.

The next parameter that the Canon looks for is "point size," the height of a typeface. There are 12 points in a pica, 6 picas in an inch. Thus, there are 72 points in an inch.

Don't confuse this measurement scheme with the unfortunate use of the word "pica" to refer to 10 pitch typefaces. Note, too, that although one sixth of an inch seems like an awkward measure, standard *line* spacing on typewriters is 6 lines per inch.

So how tall is "normal" type? Although some typesetters have a problem nailing point size to an absolute scale as it should be, "normal" 10 pitch type is about 10/72nds of an inch high, or about 10 point. Fine print, perhaps for footnotes or contracts that you hope no one will read, is about 8 point. Some headlines are about 14 point.

According to some designers, if one scheme is good, two schemes are better, and more than two would be fantastic. Canon thinks that if you have trouble with points, you might want to specify your type height in 1/300ths of an inch instead. If you don't like that, how about in 1/720ths of an inch? Of course, this means that at initialization you must specify which scheme you're using or risk obtaining some unamusing results.

The unfortunate result of this variable specifications scheme is that some heights cannot be precisely translated into alternate schemes. The point sizes specified for several of my fonts include 8.16, 8.5, and 14.4. A little mathematics will show that these point sizes started out as whole numbers in other numbering schemes (see Figure 1). This is why the hand grenade approach is useful—you just have to come *close* to the height.

Figure 1: Table of Point Conversions for Sample Canon A1 and HP Cartridge B Fonts

Points 1/72nd inch	Inches	1/300th inch	Sample Typeface
8	.1111	33.33	TMS RMN 8 (HP)
8.16	.1133	34.00	Garland S
8.5	.1181	35.42	Line Printer (HP)
10	.1388	41.67	TMS RMN 10 (HP)
12	.1667	50.00	Garland Reg, B, I
14.4	.2000	60.00	HELV (HP)

The next parameter is "style." A character is either "upright" or "italic." Note that although some dot matrix printers can italicize resident typefaces and some sophisticated laser printers can perform this same "slanting" of characters, the Canon A1, more like actual typesetting, requires a different font if you want italic. Many cartridges come with both upright and italic fonts.

The last parameter is "stroke weight." Strokes can be either regular (called "medium" by HP), light (for tiny typefaces), or bold. Bolding on some dot matrix and daisy wheel printers is created by shifting the paper slightly and striking the character again. More sophisticated bolding is possible, and any given laser font is already assigned a particular stroke weight. A different weight, particularly bold, is often available on another font on the same cartridge for variety.

So, in order to call out a typeface, you need to specify orientation, character set, typeface, pitch, point size, style, and stroke weight. Come close enough, and you win the prize.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Invoking Typefaces on the Canon

In putting together the not-quite-matched trio of printer, cartridge, and word processor, my first task was to understand how typefaces are called out. Next was to determine what was necessary to initialize the printer, then what special codes would correspond to the various parameters.

Again, the Canon manual was less than helpful. It certainly wasn't necessary to set every code in the manual, but the manual didn't identify the minimal acceptable initialization sequence.

Word Perfect 4.1, however, provided most of the answers because one of the printers it supports is the Canon A1. By examining the sequences already established, I was able to determine what factors were important—at least for initialization and for calling out certain resident fonts. Unfortunately, matching a control sequence found in Word Perfect to the appropriate explanation in the Canon manual required searching the entire manual because there is (somewhat understandably) no list of control (or escape) codes. Figure 2 lists the initialization codes and their "meanings." Note that some of these codes set incremental spacing in both the horizontal and vertical dimensions, as well as tell the printer that we're going to be using ISO codes instead of Diablo compatible codes. "ISO" is a buzzword that refers to a set of international standards and appears frequently in printer documentation today. There are totally different codes for the more limited Diablo mode.

Figure 2: Initialization Codes for Word Perfect, Canon A1 Printer, and Hewlett-Packard Font B

ESC ;	Enable ISO mode
CSI ?1;4;5;8l	Various release modes
CSI 11h	Position unit mode is SIZE (SM)
CSI ?2h	No auto form feed set
CSI 7spI	Select size unit as 1/300th inch
CSI 0'	Horizontal position absolute (HPA)
CSI 0d	Vertical position absolute (VPA)
CSI 50j	Horizontal position backward (HPB)
CSI 100k	Vertical position backward(VPB)
CSI 0t	Current horizontal position is left margin
CSI 2t	Current vertical position is top margin
CSI 10y	Typeface is #10 (Garland)—OK for TMS RMN
ESC (B	Designate graphics set DG0
CSI 22;23m	Regular stroke, upright
CSI ?0spK	Proportional pitch

Note: ESC is ASCII 27, CSI is ASCII 155, sp is ASCII 32. Other characters, including punctuation, are sent as themselves or ASCII equivalent. Release modes end with a lower-case ell, not a one. Select size unit uses an upper case eye. HPA uses the backward apostrophe. VPB uses lower case kay, while pitch uses upper case kay.

To get the system to work properly, I wrote a simple BASIC program to output the sequences so that I could experiment. I *knew* what BASIC was doing, but I wasn't entirely sure what Word Perfect was doing when various things appeared (incor-

C CODE FOR THE PC

source code, of course

GraphiC 3.0 hi-res color plots	$300
QC88 C Compiler	$90
Concurrent C	$45
Coder's Prolog in C	$45
Translate Rules to C	$30
LEX	$25
YACC & PREP	$25
Small-C compiler for 8088	$20
tiny-c interpreter & shell	$20
Xlisp 1.5a & tiny-Prolog	$20
C Tools	$15

The Austin Code Works
11100 Leafwood Lane
Austin, Texas 78750-3409
(512) 258-0785

Free shipping on prepaid orders — No credit cards

Circle Number 30 on the Reader Service Card

REAL - TIME GRAPHICS ANIMATION

For the IBM PC, XT, AT or compatible, with Color Graphics Adapter and color monitor.

With this plug-in accessory, your PC can produce graphics as fast as expensive dedicated graphics work stations. The card acts as a graphics coprocessor to the CPU. Graphics primitives are drawn by an HD63484 chip. No need to compute the adresses of individual pixels in a line, the card takes the ends of the line as input then draws the line at high speed. The line appears on the screen at a million color pixels per second.

Keep your color graphics adapter and standard RGBI monitor, this card uses both. Since your old adapter remains a part of the system, you are sure that your machine remains completely IBM compatible. Gives you all 16 colors on the screen at the same time, even with 640x200 dots.

Are you tired of having to program graphics applications in assembler and C? This card comes with an include file for Turbo Pascal. Nearly 70 procedures and functions allow you to use a high level language to write fast graphics programs. In addition to the usual lines, circles, ellipses, rectangles, 2-color area fills, etc., this package includes support for real-time animation. You can swap in a hidden screen during vertical retrace. Your animation will be clean and synchronized - no hash the easy way. Automatic checkerboard dithering is supported, giving 256 psuelo colors on a standard IBM color monitor.

Envious of your friends with Amigas and STs? This card can redraw a 21x21 3-D net graph in 1/60 sec. That is 50400 vectors per second. And the card includes support for 3-D figure drawing and rotation.

It even works with your speed up cards and fast mother-boards. No memory addresses are used, and only four bytes of I/O ports are needed, so the board won't conflict with your other graphics adapters or peripherals.

This card is available only directly from the factory. Two cards are available, one with a resolution of 640x200, the other one goes to 640x400 but requires a long persistence phosphor monitor. Both cards come with a 30 day money-back trial period. Try the card, if it doesn't perform as claimed, or if you are unsatisfied with it for any reason send it back for a full refund. The cards also come with a 180 day warranty, and include free software updates.

EHGCp2	640x200	only	$374.00	Check or Money Order
EHGCp4	640x400	interlace	578.00	30 day trial 180 day warranty (714) 261-1725
	Shipping and Handling		4.00	

E-Heart Engineering / 18103 Sky Park South #D / Irvine, CA 92714

Circle Number 67 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

rectly) on the printer. Clearly, it was necessary to separate the codes from the complex process of word processing, which was likely sending all sorts of control information to the printer. Once everything were functioning properly, Word Perfect could be finally configured.

I also periodically diverted output to my dot matrix printer, which fortunately didn't understand most of the Canon sequences, and thus was willing to output the IBM character set instead of acting on the codes. Thus, it was easy to examine most of the actual sequences sent from Word Perfect during initialization. I also examined the stream of codes sent by a popular laser utility package as it tried to select my fonts. They were not, of course, precisely the same as the Word Perfect codes.

The Word Perfect initialization sequences worked, and a great deal of experimentation with changing or omitting some codes yielded no end of confusion, compounded initially by my failure to completely reset the printer between tests. The printer would thus "learn," and what it already knew was not apparent.

In particular, it became obvious that the printer was oblivious to the fonts available on the (HP) cartridge until those fonts could be called out by software. Once any font on the cartridge had been called out, *all* fonts on the cartridge were available, by number, from the printer's switches.

The control codes in Figure 3 show how to call out this particular combination of fonts. Note that an ASCII 155 precedes most codes on the Canon, but some codes begin with the more common Escape (ASCII 27). This initialization, and these codes, will properly select the Canon's resident fonts and the fonts from HP's Cartridge B.

Figure 3: Codes to Call Out Fonts (Canon A1 with HP Font B) in Word Perfect

Font 1: HP Cart B, TMS RMN, 10 point, proportional, upright
Word Perfect Character Table: Laser-B Normal
Shift into Font 1:
<155>42<32>C<155>23m<M>
Shift out of Font 1:
<N>

Font 2: HP Cart B, TMS RMN, 10 point, proportional, italic
Word Perfect Character Table: Laser-B Italic
Shift into Font 2:
<155>42<32>C<155>3m<M>
Shift out of Font 2:
<155>23m<N>

Font 3: HP Cart B, TMS RMN, 8 point, proportional, light stroke
Word Perfect Character Table: Laser-B Footnt
Shift into Font 3:
<155>33<32>C<155>2m<M>
Shift out of Font 3:
<155>22m<N>

Font 4: HP Cart B, HELV, 14.4 point, proportional, bold
Word Perfect Character Table: Laser-B Helv
Shift into Font 4:
<155>60<32>C<155>1m<M>
Shift out of Font 4:
<155>22m<N>

Font 5: Canon Resident Courier, 12 point, 10 pitch, regular
Word Perfect Character Table: A1 Normal
Shift into Font 5:
<155>50<32>C<155>3y<155>0<32>K
Shift out of Font 5:
<155>10y<155>?0<32>K

Font 6: Canon Resident Courier, 12 point, 10 pitch, italic
Word Perfect Character Table: A1 Normal
Shift into Font 6:
<155>50<32>C<155>3m<155>3y<155>0<32>K
Shift out of Font 6:
<155>23m<155>10y<155>?0<32>K

Font 7: Canon Resident Courier, 8.16 point, 15 pitch, regular stance and weight.
Word Perfect Character Table: A1 Normal
Shift into Font 7:
<155>34<32>C<155>3y<155>2<32>K
Shift out of Font 7:
<155>10y<155>?0<32>K

Font 8: Canon Resident Line Printer (Rotated to landscape mode), 8.5 point, 16.66 pitch, upright, light stroke
Word Perfect Character Table: ASCII Line Ptr
Shift into Font 8:
<155>1p<155>35<32>C
Shift out of Font 8:
<155>0p

Notes: Numbers in < > are ASCII—this is Word Perfect's format. <M> is Word Perfect's code for turning proportional spacing on; <N> is the code to turn it off. See Figure 1 for initialization codes because these codes assume initialization to proportional upright font—anything else must be shifted into and out of.

Setting Up Word Perfect

Now that I could select fonts from BASIC, the only remaining task was to configure Word Perfect. Because there are only 8 fonts available at one time in Word Perfect, you must choose carefully. I experimented for a while and found that WP was perfectly capable of bolding most of the fonts (using key F6), so I didn't waste any of the 8 fonts by specifying a bold face.

In addition, WP came configured for the Canon with some double wide and/or double tall fonts that look terrible. The Canon has the built-in ability to scale fonts to double size in width, height, or both. Unfortunately, it does this by simply blowing up the character, and the resolution thus becomes twice as crude—jagged edges are apparent. I find this unacceptable although many people seem highly enamored of the effect.

Two other built-in effects work well. The Canon can reverse (i.e., print white letters on a dark background) and shadow (print a closely spaced dot pattern in the background). Since these effects work on all typefaces, it seemed more intelligent to assign them to macros rather than take up any precious font assignments. See Figure 4.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Figure 4: Useful Macros for Canon A1 and Word Perfect

RON.MAC	(Turns reverse on):	<155>7m
ROFF.MAC	(Turns reverse off):	<155>27m
SON.MAC	(Turns shadow on):	<155>5m
SOFF.MAC	(Turns shadow off):	<155>25m

Here's an explanation for configuring RON.MAC: Type Ctrl F10 to invoke the macro definition. Enter the name "RON" and press return. Press Ctrl F8 to invoke the Print menu. Press "A" to select "Insert Printer Command." At the "Cmd:" prompt, type the symbols for the macro, including the "<" and ">," and be sure to use the lower case em. Press return to end the command sequence.Press return again to exit from the print menu. Press Ctrl F10 to end the macro definition.

Whenever you want to start reverse type in your document, press Alt and F10 and enter "RON" for the macro name. Similarly, invoke the "ROFF" macro to stop the reverse printing. If you use reverse print in headlines, you'll have to experiment with margins and margin releases to prevent bolding the margins as well as the headlines.

One serious problem remained. Right justification simply didn't work correctly. Lines did not come out even at the right edge, and the spacing of the rest of the line didn't quite "look right." Logically, the inability to properly justify text is an immediate tip-off that Word Perfect doesn't have the correct typeface stored for the one that's actually being called out.

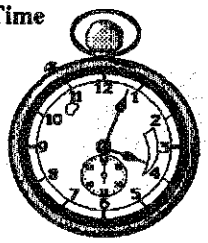
Remember the discussion about proportional spacing? If Word Perfect doesn't count micro-spaces the same way the printer does, it won't throw the proper number of micro-blanks into a line. Since every typeface differs somewhat in its internal proportions, it must be precisely identified to Word Perfect.

An examination of the printer drivers available to Word Perfect shows that there are typefaces associated with them. In particular, the HP LaserJet (with B cartridge—many cartridges are available on the list of printers) can be selected as a printer and installed, and at this time the full set of "B" typefaces will then be available. Canon refers to these as "Laser-B Normal," "Laser-B Italic," "Laser-B Footnt," and "Laser-B Helv."

So the last task is to re-install the Canon A1 printer, assign the appropriate HP typefaces to the 8 fonts, and check the control codes in each font assignment. Although Word Perfect will let you change the proportion assignments, because the HP fonts have already been "measured" and put into the large pool of available fonts, this is unnecessary. The only trick is to install the HP printer temporarily to make the typefaces available. Here's how it's done: Although a lot of what you need is in a utility program called "PRINTER.EXE," you should start from within Word Perfect itself. Press Shift F7 to invoke "Print," and then choose #4, "Printer Control." Next choose #2, "Display Printers and Fonts." This will show you each printer you currently have installed and which typeface has been specified for each of the available 8.

Return to "Printer Control," and this time choose #3, "Select Printers." This tricky little section shows you the printers already selected at the top of the screen and the correspondence between printer number and printer at the bot-

Along With Your Computer, Your Time is the Most Important Thing You Own. . .So Why Waste It?



Quilt Programmer Productivity Tools will help you manage your software projects and get control of your time!

SRMS™
Software Revision Management System

SRMS is a full featured revision control system that allows programmers, system managers, and librarians to get control of software projects. SRMS stores every version of a source program in a single ASCII file, with the ability to restore each and every version easily and quickly. Only changes (deltas) between versions are actually stored, so disk usage is drastically reduced. Each version is stored with a unique Version Identification String and unlimited user supplied comments, making identification and documentation tasks easy. SRMS supports projects written in any language, and allows you to use your current editor and compiler without conflict. Multiple programmer projects are offered added protection through the use of "lock" files controlling access to each library. SRMS also incorporates the use of "classes" to bind various versions of different modules together for reconstructing a particular software system release. DOS directory pathnames and environment variable support also aid you in your work. Full typeset documentation with tutorial and project management chapters.

SRMS ..... $ 125.00

QMAKE™
Automatic System Generation Utility

QMAKE is a powerful and flexible utility that can be used to control the rebuilding of simple or complex software systems. QMAKE relieves the software developer of the task of remembering which modules of a system need to be rebuilt based on recent changes, how to rebuild them, and in what order to rebuild them. Once you specify the relationships and commands necessary to rebuild your system, and place them in a control file, called a makefile, QMAKE can be used to rebuild your system through the invocation of a single command.
QMAKE will work with most compilers, assemblers, and linkers. It supports full macro definitions, UNIX makefile compatability, recursive invocations, and interpretation of even the most complex dependencies.
QMAKE will also interface with our Software Revision Management System providing you with a complete set of productivity tools to handle any of your project requirements. Full typeset documentation with tutorial.

QMAKE ..... $ 99.00

SRMS + QMAKE ..... $ 199.00

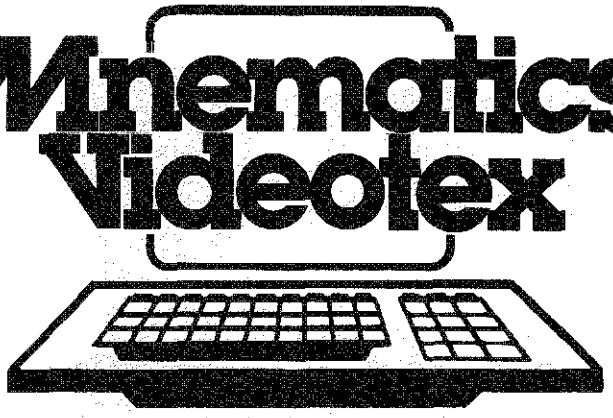
QUILT COMPUTING
7048 Stratford Road
Woodbury, MN 55125
(612) 739-4650

Volume Discounts and Dealer Inquiries Welcome

Circle Number 46 on the Reader Service Card

The people you've been wanting to meet are online with...

Mnematics Videotex



Fast! Friendly! Fun!
At a price you can afford!

Access numbers in over 800 cities and 120 countries.
Over 50 SIG's, Databases, Games, Realtime Chat and more!
300/1200/2400 baud.
Mnematics Videotex, Sparkill, New York 10976-0019

Call 1-800-322-3633

(914-365-0184 in N.Y. State)

Circle Number 65 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

tom of the screen. Although you may have defined a dozen printers to the system, you will have only one printer associated with number 1, and this is the printer that Word Perfect will use until you tell it differently (under Shift F7, #3 Change Options).

Press PgDn for additional printer definitions, where you'll see a list of the many printers supported by Word Perfect. (You'll need to have your printer diskette handy to get the list.)

You'll find that the HP LaserJet is listed many times—each time with a different cartridge. Find the number that corresponds to the cartridge you're going to use. Select the main printer number you want, perhaps number 2, using the arrow keys, and type in the number of the HP LaserJet/cartridge. Then tell Word Perfect what printer port you're using (LPT1, COM1, etc). You've just assigned a new printer. The important thing, though, is that you've just made the stored font definitions available to *any* printer that you choose to install!

Repeat this procedure, using a different printer number, but this time select the Canon A1 from the list. Now you've got two printers installed, but neither one of them is correct!

At this point you can browse through the default settings that we'll discuss below. Notice the enormous difference in the codes used by Canon and HP to accomplish the same tasks, despite the fact that the underlying print engine is the same (they're both Canon).

Now go back and assign the Canon to whatever permanent printer number you'd like, presumably printer number one.

Now run Word Perfect's utility program PRINTER. It offers you

1. General Information and Help
2. Explanation of Special Codes Used In This Program
3. Printer Definitions
4. Character Tables
5. Sheet Feeder Definitions

Read the general information and information about special codes in Sections 1 and 2. This makes it easier to enter information later.

Select number 3: "Printer Definitions." On the list of printers you'll see the Canon A1. Type "D" for rename, and name it something like "Canon A1 with HP Font B"—not very catchy, but that keeps it from getting mixed up with the standard A1.

Now type "B" for edit. You'll see a list of features that can be edited, including printer initialization, fonts 1-4, fonts 5-8, and character tables. Choose #1, "Printer Initialization," and make sure the codes match those in Figure 2. Note that there are so many codes that you must scroll them left to see the remainder.

Now choose #8, "Fonts 1-4," and enter the correct codes to shift into, and out of, the first four fonts. You can use the codes in Figure 3, or you can choose those codes appropriate to your particular cartridge. Do the same with #9, "Fonts 1-5."

Finally, you need to assign the correct typeface from the list to each of the 8 available fonts. If you fail to perform this last task, it will become obvious when right justification fails. Word Perfect will be using the wrong micro spacing.

From this same menu, choose "A," "Character Tables for Fonts." You'll see a list of character tables assigned to each of the 8 available fonts. Choose Font 1, and you'll see a list of available character tables, including the Canon built-in set, the HP set, and any other set that you may have previously made available. If the first font was configured for the normal TMS RMN character table, for example, you would choose "Laser-B Normal" from the list.

Keep in mind that in assigning fonts, you must tell Word Perfect what control sequences to send in order to shift into, and out of, each font, *as well as* the character table to use for that font.

For your own interest, return to the main menu and examine the character tables (choice #4). Note that the extensive amount of configuration possible makes all of these menus somewhat confusing. "Character tables" on the main menu lets you examine the tables and change the individual character definitions while "Character Tables" on the "Printer Definition" menu lets you assign tables to fonts.

Choose one of the character tables from the list, and you'll see the ASCII value of each character, a screen display of the character, the ASCII value of the character that will be printed when this item is invoked, and the width of the character in 1/300ths of an inch. Note that many typefaces do not support the IBM graphics characters (such as the box pieces), so although they will display, they will print as ASCII character 32—the blank. This is particularly unfortunate because many programs make use of these characters to print nice forms. Word Perfect's very handy drawing routines are rendered useless with one of these fonts, which include HP's Cartridge B.

Although installation is not easy, using these codes and techniques, anyone should be able to "mix and match" to use the Canon A1 printer with virtually any HP font cartridge to create high quality work with Word Perfect. But unless you're prepared to assign all the micro spacing yourself, be sure to choose a cartridge that's already supported on its native machine.

Problems for the Future

Our journey through the intricacies of this particular laser printer, cartridge, and word processor combination has some important implications for future development.

As we use laser printers, we must either learn to work around the limitations, such as envelopes and labels, in order to obtain the quality, or we must wait until better hardware comes along. We're also faced with installation of software that is now considerably more complicated and problematical. Users will need professional help for system configuration, and dealers and VARs should be prepared to guarantee that all components will function properly together. Although many routine tasks have become far easier on computers today, the additional complexity and sophistication of today's hardware and software diminish this ease of use.

For those of us who write software, laser printers introduce some nasty complications. It's no longer sufficient to allow the user to simply specify a few printer codes for regular and condensed print at installation of a custom package. We now need to know *at the least* whether the font that will be used supports the graphics character set, what the entire initialization sequence might be, and potentially, a long series of codes to select each typeface. In addition, most of us are not equipped to deal with proportional typefaces on reports—this would lead to unnecessary chaos. As I write this, I'm thankful that I have this marvelous equipment, but I'm equally relieved that none of my clients is using a laser printer—yet. □

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

SPEEDWARE™

A DIVISION OF AMERICAN AFFORDABLE SOFTWARE CORPORATION

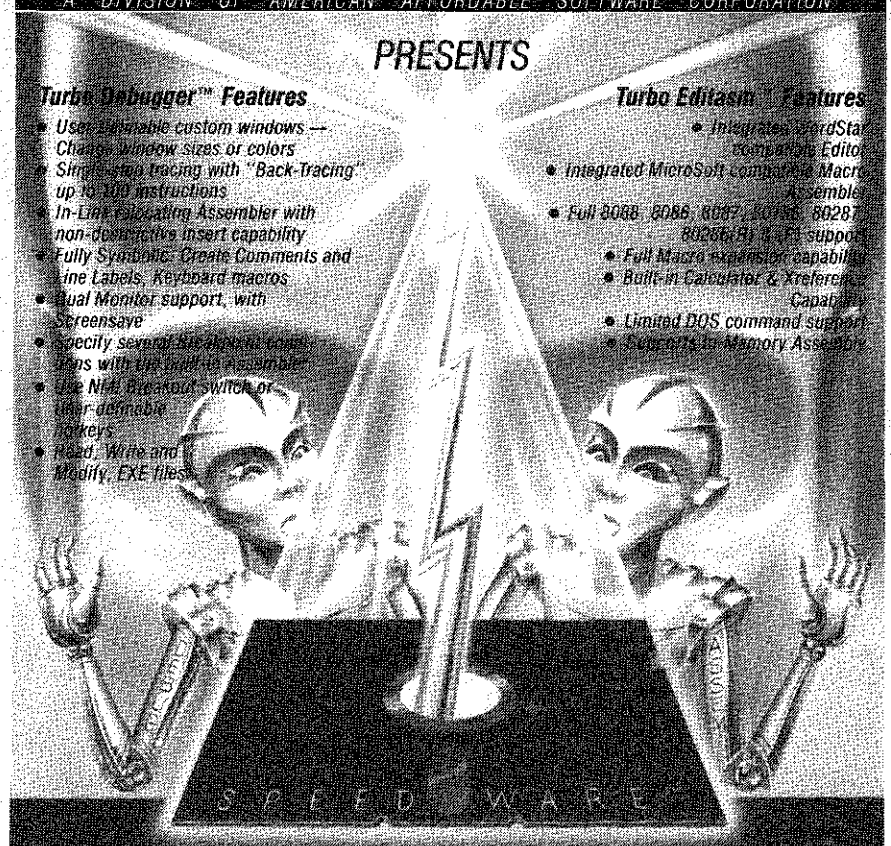
PRESENTS

Turbo Debugger™ Features

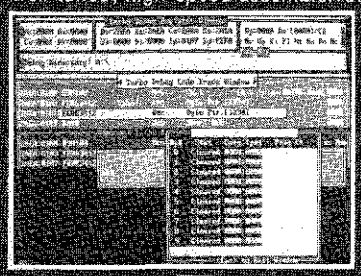
- User-definable custom windows — Change window sizes or colors
- Single-step tracing with "Back-Tracing" up to 100 instructions
- In-Line Executing Assembler with non-destructive insert capability
- Fully Symbolic: Create Comments and Line Labels, Keyboard macros
- Dual Monitor support, with Screensave
- Specify several Set-Aborts conditions with the built-in Assembler
- Use Null Pointer/out switch or user-definable Tokens
- Read, Write and Modify .EXE files

Turbo Editasm™ Features

- Integrates WordStar Compatible Editor
- Integrated MicroSoft compatible Macro Assembler
- Full 8088, 8086, 8087, 80186, 80287, 80286/87 & EGA support
- Full Macro expansion capability
- Built-in Calculator & Xreference Capability
- Limited DOS command support
- Supports Expanded Memory



To maximize your programming potential order the ultimate "Turbo Team" today — the first well-tuned, high-speed assembler/debugger for the IBM P.C. and compatibles family.



$99.00 Turbo Editasm™

$89.00 Turbo Debugger™

THE TURBO TEAM™

Inside California (800) 334-3664 • Outside California (800) 443-6190

$99.00 ☐ Turbo Editasm™
$89.00 ☐ Turbo Debugger™

Name _____
Shipping Address _____
_____
City _____
State _____ Zip _____
Telephone _____

Amount Enclosed _____
Payment VISA MC BANK DRAFT CHECK
Credit Card Exp. Date ___/___
Card # _____

CA Residents add 6% sales tax.
Outside USA add 10% and make payment by bank draft payable in U.S. dollars drawn on a U.S. bank.

Dealer and Education Discounts Available.

Speedware™
9719 Lincoln Village Dr., Ste. 303
Sacramento, CA 95827

Circle Number 72 on the Reader Service Card

Moving?

Please send us both your old and new addresses, with Zip Codes. If possible, include the address label from a recent issue of *Programmer's Journal*. Allow three to four weeks for processing.

Programmer's Journal
P.O. Box 3000
Dept. EE
Denville, NJ 07834

New Version 2.0

Complete C Programs in Half the Time, with *Instant-C*™

You can create programs much faster with Instant-C than with conventional programming tools. How? Because Instant-C is a high-performance interpreter, there are **no compile or link delays.** Change your program, then test it immediately. No matter how large your program, the turnaround time is just seconds.

"Instant-C means instant gratification."—*PC Magazine*, **Editor's Choice** for best C interpreter.

Powerful **source-level debugging** saves your time. Conditional breakpoints, single-stepping by statement, source code backtraces, data monitoring, and many other debugging features make it easy to wipe out bugs quickly. Direct execution of any statement or function makes testing a breeze.

"The resulting debugging and testing capabilities are fantastic and the detailed trace/debug/display commands make it easy."—*The C Journal*

Instant-C checks pointer references for reasonableness, and checks that array indexes are within declared bounds. This **run-time checking** stops your program as soon as errors occur, for easiest debugging.

Not only does Instant-C help you quickly change, test, check and debug your code, but it runs your program **fast enough for real-time** applications.

"It is much faster than any of the other products mentioned and was the only one able to complete the standard SIEVE in a reasonable time. Clearly, this high speed allows much more complex problems to be attacked with Instant-C than with any of the other products discussed."—*Computer Language*

Immediate feedback and precise diagnostics make Instant-C great for learning C. Full K&R and the ability to **link compiled object code and libraries** (Lattice and Microsoft) makes Instant-C compatible with your existing programs.

Instant-C makes all parts of the programming task as fast as possible.

"Clearly, Instant-C is the performance champion."—*PC Tech Journal*

Version 2 works with MS-DOS and PC-DOS, and has a full 31 day **money back guarantee.** Instant-C is only $495. Order today! Call or write for full information.

Rational Systems, Inc.
P.O. Box 480
Natick, MA 01760
(617) 653-6194

Circle Number 20 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Genifer: A Powerful dBASE Applications Generator

by Paul Barkley

GENIFER 1.0

Bytel Corporation
1029 D Solano Avenue
Berkeley, CA 94706
(415) 527-1157
$395
Not Copy Protected

If you write code in dBASE and don't use some sort of application generator, you're living in the stone age. Run out today and pick up a copy of Genifer and spend a couple days with it. At the end of that time, you'll either thank me profusely for making you much more productive, or you'll decide that while they're on the right track, you can do better if you "roll your own." Let's see why.

An application generator is a piece of software that asks you some intelligent questions about the design of your system and then produces reams of perfect code, in a particular language, so that you don't have to write that code yourself. If you make your living writing applications to make your clients more productive, it's about time you thought about making yourself more productive.

Besides getting the job done far more quickly, an application generator produces uniform code that is easy to modify at a later time, and it completely removes the need for debugging during application development. Well . . . *if it works properly* in the first place, you won't have any bugs. Otherwise, you'll get the same bug in application after application. Because the code structure is identical between applications (only the data forms change), you can put every effort into building an optimal program form—a template as it were. Once the template is clean, all applications generated from it will be clean.

Genifer (1.0) is a decent application generator that turns out very good dBASE code, and with some limitations can create code to be compiled with Nantucket's Clipper or WordTech's dBASE III Compiler. Like all application generators, it has certain necessary features.

Data Dictionary

First, it runs from a data dictionary. You name and define all your files, telling the system what the fields are like. Besides the normal dBASE information like type and size (and decimal places), Genifer lets you specify a default value, a picture (using the dBASE extensive format conventions), and a validation procedure. Fields can be validated against a range or a list of appropriate values or can be looked up in a file. These features form an integral part of the file management module discussed below.

One nice feature of Genifer is that it will import this information from dBASE files that are already created, so if you're converting an application, you don't have to enter the information already known to dBASE. You can also modify the structure of a file, as in dBASE, in case your old format needs revision.

Menus

Next, you create your menus. Genifer will ask you if you already have a form and will use that as a starter. Either way, Genifer puts you into your favorite word processor or editor, and you can create a menu page, with boxes or anything else that your word processor can handle. It's up to you to make the menu coherent by listing the choices and the appropriate input next to them. You can use numbers or letters for your user menu prompts, and at the bottom (usually) you type a line that says, "Enter your choice" or whatever wording you prefer. Next to that you'll use Genifer's special input character, which is an underline. If you type two underlines on the screen, Genifer will know to do a two-character "GET" at that point. If you don't like the underline character, Genifer will let you customize it to something else.

When you exit from your editor, Genifer asks you for the choices that you will accept and the program name that it should chain off to if that choice is entered by the user. By giving Genifer this information, you allow it to write the code for a single menu program that branches appropriately. You repeat this procedure for each menu.

File Maintenance

In every application, you must be able to maintain files. There are only so many things that anyone can do to data in a file: add, delete, or change records, or move through the file with find, next, previous, top of file, bottom of file, and skip a number of records. To be consistent a good application system will use one program that will allow all this maintenance on one file and then contain (or appear to contain) an identical program for all other files. The only thing that will differ is the structure of the particular file itself.

Genifer helps you create a file maintenance system by asking you to paint a screen with your editor, just as you created a menu. When you come back from the editor, you must tell Genifer what field is associated with each of the underlines that you left on the screen. Genifer will do the rest because it already knows how to write a file maintenance program in general, and it has all the data it needs about this particular file from the data dictionary (remember telling it what default values you wanted and how to validate a field?).

One problem with generated file maintenance programs is that as soon as you start creating single file programs, you'll get a client who needs to be able to update more than one file at the same time. You need the capability to do the same tasks, on both data files, but to be able to switch easily between them.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Genifer gets around this very adroitly by letting you create second or third screens that are linked. In addition to the file maintenance commands I've already mentioned, the user can also "Zoom" if there's another screen "below" this one. So, you can enter a master record, zoom to the transaction file, and enter a number of related transaction records. Information can be carried over, using a copy feature (like "SET CARRY ON"). After entering all the related transaction records, the user can then quit from that screen and be taken back to the first screen.

During the definition of these screens, you're asked if you want the screen to be erased before the screen displays. Usually you specify "yes" for the first screen and "no" for the second so that the top of the screen with the master record stays in place while the user enters the transactional data. The Genifer tutorial, which is very good, has an example with three partial screens linked together.

Obviously, there are data entry procedures that won't fit into this format, but Genifer has features sufficient to make this handle most of what needs to be done in the typical application.

Reports Are Easy

The third major area of an application, after menus and file maintenance, is report generation. This is where Genifer really shines. One of the ridiculous limitations of most report generators today (like the one built into dBASE) is that they can report from only one file. dBASE creates nice looking reports very quickly if you're willing to accept this limitation. A typical application system might have a couple of one-file reports, but most reports will require at least a look up into a secondary file.

To create a report, Genifer asks you for the primary and secondary data bases and the field that they should be related on. Genifer then puts you into your editor again, where you lay out your report format. As before, Genifer is polite enough to ask if you'd like to start with a template for your reports. This is important if you want to minimize the amount of time spent typing headings into your editor. It's also important so that all your reports have uniform headings, with no odd spellings or placement.

Genifer accepts anything you'd like to put into the report. Fields are specified as usual with underlines, and calculated data is placed that way, too. Genifer also accepts special codes for the date, time, page number, and number of records to make it easy to include this common report information.

When you return to Genifer from your editor, it then scans your form and asks you line by line to identify what you've created. In this way you specify lines as belonging to the report header, report footer, page headers, page footers, break headers, break footers, or detail lines. When you specify break headers and footers, you're asked for the break level. Any place that you have an underline, you're asked whether it's a field, a total, or an average, and for the fields you're asked to name them and specify whether they're from the primary or the secondary file. This seems unnecessary but covers duplicate field names where the field contents might not be the same in both related records because it wasn't the field that the files were related on.

What a sense of freedom to no longer be confined to a single detail line, or a single break header or footer! You can make the layout anything you want.

The only price you pay for this is that it takes a few minutes longer than in dBASE to create a very simple report, but this is more than made up when you try to create something more complex.

QuickBASIC just got quicker with

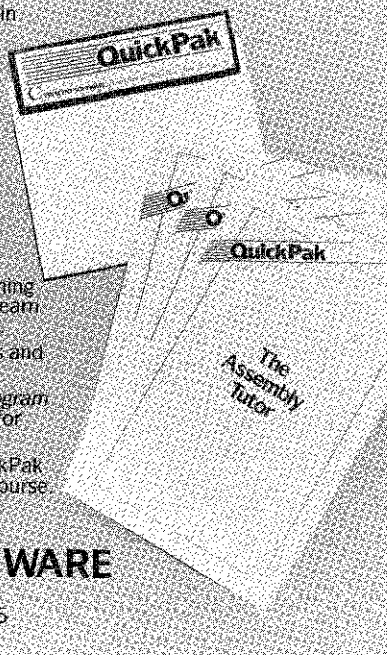
QuickPak

QuickPak is a superb collection of enhancements, subroutines, and instructional material designed to help you get the most out of programming in BASIC.

- Powerful assembly language routines to give your programs more speed, more power, and full access to DOS and BIOS services. SORT all or part of a string array with one command! FIND any string or sub-string within an entire array regardless of capitalization — accepts wildcards. READ directories into your programs from any drive or path. READ/WRITE disk sectors — create your own DOS utilities! CLEAR or SCROLL any part of the screen. MANY, many more programs included.
- Professionally written QuickBASIC routines and functions. Powerful input routines for text, dates, and numbers. Menus, scroll bars, date/time functions, and much more.
- The Assembly Tutor — a complete guide to learning assembly language *from a BASIC perspective*. Learn how to create your own routines and extensions.
- Tips and Tricks book — packed with clever ideas and techniques to help you be a better programmer.

You get all this, all of the source code for *every program included*, and a thirty-day money back guarantee for only $69.00.

No royalties are required for using any of the QuickPak routines in your programs. Not copy protected, of course.



by

CRESCENT SOFTWARE

64 Fort Point Street
East Norwalk, CT 06855
(203) 846-2500

QuickPak requires Microsoft QuickBASIC or BASCOM, DOS 2.0 or higher. Visa, M/C, C.O.D., or checks accepted.

Circle Number 83 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

C
DYNAMO
FREE!
PCUNIX™
(WITH ANY PURCHASE OF $250 OR MORE. LIMITED TIME OFFER)

LIBRARY: *325 fully tested functions screen handling/graphic, cursor/ keyboard/data entry, string handling, status and control, utility/DOS/BIOS/ time/date, printer control, 4 disks* $129.95

WINDOWS: *The most powerful windowing package available, overlays, borders, pop-up menus/ help windows, zap instantly on/off screen, status lines, horizontal/ vertical scrolling, color control/ highlighting, word-wrap, files to windows, keyboard to windows 3 disks* $129.95

SUPERFONTS FOR C: *Dramatic, high impact screens. Use our character & image libraries and functions, or create your own. Font & Function LIbrary* $49.95

B-TREE LIBRARY: *Fast indexing B-tree. 16 million each: keys and records, unlimited keys per file, variable length records. PC-DOS, UNIX, XENIX, CP/M.* $79.95

ISAM DRIVER: $49.95

COMPATIBILITY: *PC/XT/AT; Full K&R, CI-C86/De Smet/ Lattice/ Microsoft 3.0/Aztec/others.*

VALUE: *All source code, No royalties. Best documentation available.*
No matter what else you have, you need these.

C-TERP: *Simply the best C interpreter available. See C Journal Summer 1985*
C-TERP (specify compiler) $299.95

PCUNIX: *Multi-tasking, networking multi-user. With source code* $99.95
™Wendin Software

COMBINE AND SAVE!
C LIBRARY plus C WINDOWS
BOTH for only $179.95
+ SUPERFONTS FOR C . $199.95
+ B-TREE and ISAM $299.95
(A $440 VALUE)
C LIBRARY plus C WINDOWS
+ SUPERFONTS
+ C-TERP (a $610 value) $459.95
All (a $740 value) $549.95

Entelekon

12118 Kimberley, Houston, TX 77024
713-468-4412
VISA MASTERCARD CHECK COD

Circle Number 9 on the Reader Service Card

So there you have the essence of Genifer: a data dictionary that drives menus, file maintenance programs, and reports. What more could anyone want?

Nitpicking

Well, I have a few complaints. It's obviously impossible to put together a tool that writes programs that suit everyone, and while I wouldn't hesitate to recommend Genifer, it may not be right for everybody in its present form.

A couple years ago, in self defense, I wrote an application generator for dBASE code, so I have some pre-conceived notions of how one should be designed.

I like to see the menus handled differently. It makes more sense to me to put the menu structure into a file so that *all* the menus in a system are present at one time. The display presentation of menus to the user should always be the same, and writing each menu with your editor is a good way to mess that up and is a waste of time as well. It's easy to slightly reword the prompts on each screen, for example.

Some application generation schemes use this menu file to actually drive the menus when a user is on the system, but reading the choices out of a file slows things down. Thus, while I applaud Genifer for actually creating the hard code from the menu specifications, I'd like to see this done from a single menu program skeleton that the user could modify *once* for any application. All menus would use the same skeleton. Better yet, all menus should run from one master menu program instead of from separate programs.

Another problem with Genifer's code is that it was designed strictly as dBASE code; if you want to compile it, you run into some vexing problems. Genifer addresses this in the manual, and it appears that WordTech's compiler will handle the code with little problem.

If you compile with Clipper, though (which I think is the much better compiler), it becomes obvious that there are some design weaknesses that are going to make life unpleasant. Genifer code is designed so that when a menu calls a file management program (let's call it an FMP for short), it does so by setting the procedure name to the FMP's name and then executes a "DO" to that procedure. Inside the FMP it starts by naming itself as the procedure and then calls other procedures as necessary. Figure 1 shows briefly what it looks like.

```
MENU PROGRAM
        ...
        set procedure to FMP
        do FMP
        ...

FMP PROGRAM
        procedure FMP
                ...
                do x
                do y
                return

        procedure x
                ...
                do y
                return

        procedure y
                ...
                return
```

Figure 1: Genifer's Unusual Structure for File Maintenance Programs

I could never understand why anyone would want to use procedures in dBASE code since I always set them up as a group of external routines. Thus, if a procedure were called from inside a loop, dBASE had to open that external file every time the procedure was called, and performance degraded to an unacceptable level.

Obviously, those with more byzantine minds discovered that you could place the procedure inside the same file, and even call it by the same name as the program, and overcome the performance problem. I'm not the only one who thinks this is a peculiar idea. Clipper can't handle this either and complains about a "symbol redefinition error" when it compiles.

This illustrates an even more fundamental problem for those who want to generate Clipper code. Genifer creates a different special file maintenance program for each occurrence in your application system. If you want the user to be able to manipulate 5 files, you get 5 programs. Clipper must then compile all 5 programs together, which makes your code grow alarmingly, when a better design would have been to use a single FMP that was parameter driven for *any* file in the system. It could simply read your screen format file and act accordingly (mine does that nicely).

Worse, while you're making up new names for the FMP procedure file, and changing the generated menu program, you find that each of the FMPs in your system has a set of the same procedures (labeled "x" and "y" in our schematic). These all have to have unique names, or, understandably, Clipper doesn't know

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

what to hook where. In most cases, all your total application *system* needs, though, is one of each of these procedures since many of them aren't unique (some procedures only put error messages on the screen, etc.). Some, of course, are specific to the file being manipulated, which comes back to the fundamental pro-dBASE/anti-Clipper design.

Clipper also complained about a line in the FMPs that said "pack&num" and said it would ignore the remainder of the line. Changing the code to "xpack&num" throughout solved that problem although I was surprised to see something like this in what should be "clean" code.

All these annoying Clipper problems are pretty ironic since Genifer itself was written in Clipper! That suggests that many of these issues will probably be addressed in future releases.

So, now that I've spent all this time complaining about how Genifer doesn't work that smoothly with Clipper, why do I think you should go out and buy the product anyway? First, if you work in dBASE these Clipper problems are moot. dBASE systems work great.

Secondly, it contains the essence of what you're going to *have* to use in the next couple years if you want to stay competitive. It's time to learn an application generator, and this one is good.

Thirdly, every program is well written, at least according to dBASE design criteria, and you can learn a lot about good program structure. I thought *my* reports were pretty clean until I saw what Genifer produces—it's enough to make you cry. By deftly calling routines to produce detail lines, breaks, headers, and footers, Genifer lays out the form of the report in startling clarity, and there are no break screwups with getting the wrong record's information into the break header or footer as so often happens with complex reports written and rewritten by hand. Seeing the code, and working with it, is far more valuable than listening to some wizened old codger drawing FORTRAN flow charts on a blackboard.

No matter what you charge per hour for your time, you'll make it back quickly by spending a couple days with Genifer and seeing, at least in general form, the shape of tomorrow. □

PolyMake *The Leading Make Utility*

Are you still using a prehistoric Make? Now, step up to PolyMake, the most powerful and flexible Make utility available for programmers using MS-DOS. PolyMake is like an intelligent assistant that remembers how to rebuild a program when you change one part of the program. PolyMake will automatically invoke your compiler, assembler, linker, librarian or other tools to update a single program or entire software systems when you type — MAKE. PolyMake comes with built-in rules for rebuilding programs but you can also teach it new rules so you don't have to remember the file dependencies in your program. Advanced programmers prefer capabilities like fully recursive makefile processing and the ability to invoke PolyMake with special "flags" and macros that automate a whole range tasks. New Make users appreciate the Step-By-Step tutorial and intuitive commands. Handles source files written in any language. Requires DOS 2.0 & higher. Compatible with LANs, the IBM PC, XT, AT and other MS-DOS PCs. **For complete details write for the POLYTRON Programmer's Catalog.** **$99**

PVCS *The Most Powerful & Flexible Source Code Revision & Version Control System.*

The POLYTRON Version Control System (PVCS) allows programmers, project managers, librarians and system administrators to effectively control the proliferation of revisions and versions of source code in software systems and products. PVCS is a superb tool for programmers and programming teams. (A special LAN version is also available.) If you allow simultaneous changes to a module PVCS can merge the changes into a single new revision. If changes conflict, the user is notified. Powerful capabilities include: Stores and retrieves multiple revisions of text; Maintains a complete history of revisions to act as an "audit trail" to monitor the evolution of a software system; Maintains separate lines of development or "branching"; Provides for levels of security to assure system integrity; Uses an intelligent "difference detection" to minimize the amount of disk space required to store a new version. Requires DOS 2.0 or higher. Compatible with the IBM PC, XT, AT and other MS-DOS PCs. Single User version $395. 5-station LAN version $1,000, add $500 for each additional 5 stations. **$395**

TO ORDER: VISA/MC 1-800-547-4000 Dept. No. 306; Oregon/Outside US, 503-684-3000 Send Checks/POs To: POLYTRON Corp. 1815 NW 169th PL, #2110, Dept. No. 304, Beaverton, OR 97006

POLYTRON
High Quality Software Since 1982 ®

Circle Number 26 on the Reader Service Card

PROGRAMMER'S JOURNAL

Famous Cartoons Reformatted as T-Shirts

See inside back cover for details.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Advanced Trace86: A Debugger and More

by Peter Feldman

Advanced Trace86 2.3

Morgan Computing Company
P.O. Box 112730
Carrollton, TX 75011
(214) 245-4763
$175

Writing in assembly language has always been considered very difficult, and IBM and MicroSoft's DEBUG.COM lacks many features that could help a programmer test, debug, and run an Assembler program as it is developed.

AT86 (Advanced Trace86) from Morgan Computing not only outshines DEBUG as a debugging tool, but it's also an excellent assembly language teacher for the beginner and a machine code interpreter/compiler as well. AT86 can also be coupled with the Periscope™ debugging board (see *PJ*, Vol. 2, No. 6) for even more power.

AT86 works with the 8088-8086 and the 80286 chips and with their corresponding math co-processor chips.

The program is supplied on a single disk with a 136-page instruction manual and requires 192K RAM, one or more disk drives, and either a monochrome or color monitor.

AT86 can be loaded and run as either a stand alone or memory-resident program. Variable buffers for symbol files and command stack space can also be specified at loading time.

AT86's initial display is similar in many ways to DEBUG's, with the exception that all registers and flags are permanently displayed on the first two lines of the monitor. The third line consists of a status section with a listing of the relevant commands for the current of the six screen modes available. Let's go through the six modes and explore the operations in each.

1. The Command Mode

From this central point in the program, files are named, loaded, examined, altered, and written back to disk via the other modes available. One of AT86's handy features is that information acquired in one mode can be shared and manipulated in other modes.

2. The Display Mode

We begin by loading a file and viewing its contents using the Display mode. The filename can be specified from the DOS prompt as a command line parameter or via the familiar DEBUG "N" and "L" commands. From the command mode, pressing the "D" and <Enter> keys will switch to a full screen display of the file's contents. At first glance, this display seems quite similar to DEBUG's, but the differences quickly become apparent.

The cursor's default position is at CS:0100. The Tab key toggles the cursor position between HEX and ASCII sections. The arrow keys on the numeric keypad provide full screen access to any point in the file.

Typing Ctrl A displays a 500h byte portion of the file in ASCII.

Ctrl O toggles overwrite mode, with which you can modify data in both the ASCII and HEX displays. Blocks of memory up to 64K in size can be marked and moved as if you were using WordStar on a text file.

Memory can also be displayed in both word and double word formats. For example, typing "DD 0000:0000" in command mode brings up a display of the first 320 bytes of DOS's memory showing interrupt vectors in the handy segment:offset address mode. To trace the interrupt vector, place the cursor on the address you're interested in and type Ctrl F. This will unassemble the instructions beginning at that address. Each press of the down arrow will unassemble another instruction.

3. The Assemble Mode

AT86's Assembler Interpreter/Compiler mode provides an excellent way to try out new assembly modules before incorporating them into a larger program. It's also a handy way for beginning assembly language programmers to try out sample code to see what happens (a reset button is strongly recommended).

Typing an "A" at the command mode prompt brings up the assemble mode. Unless otherwise specified, the current CS:0100 address is the beginning address point. DEBUG of course offers a similar "A" mode, but AT86 takes this function much further, acting very much like the BASIC interpreter, but creating a complete .COM program in the process. BASIC programmers will find the assembler facility quite familiar since many commands accepted by the BASIC Interpreter, such as FILES, EDIT, SAVE, RUN, DELETE, LIST, LOAD, etc., are recognized by AT86.

You can insert both comments and labels during the "A" function. The "interpreter" handles jumps to as yet unreferenced labels completely in stride, marking such addresses with a "U." If the program is later traced with an illegal address reference, AT86 prints an error message and gives you an option to continue (ignore the current command) or abort the trace. Once a prereferenced label is inserted in the code, AT86 immediately substitutes the label's name and address in all the proper places.

Each line of code entered in the assembler function is limited to 56 characters, including labels and comments. Neither labels nor comments are allowed on lines of their own; each must contain at least a NOP to keep AT86 from deleting it from the saved program. The command "SAVE progname" saves the assembled programs with their embedded labels and comments. A .COM extension is assumed. Pre-existing .COM and .EXE files can be loaded and reworked with comments and labels replacing some of the jmp instructions. AT86 permits the creation of distinct program information files which can be

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

stored with any name you choose. A special header is created for the work file, which contains information about where and how to load the label\ files.

Inserting variables into a listing is simple. AT86 supports the pseudo op "DB" and "DW" commands by creating a special DS section 32K above the current value of the CS register. As the variables are typed into AT86, the names and assigned values are stored in the DS register. To view the variable names, type "d/v" from the command prompt. Type "edit varname" to edit the contents of a variable. All the current variable contents can easily be dumped to the screen with the "dbds;0" command. Only one line's worth of data may be assigned to any one variable, but dummy variables can be assigned to store longer strings, which may be placed in contiguous sections in the data segment. The DUP operator and the "?" character cannot be used. All variables of unknown content must be initialized as "varname db 0."

To preserve variable db values in a file, a text file may be created and read into AT86 with the "n<filename" command. This file can be used to generate a complete .COM file if desired.

Once a commented and labeled program is generated, AT86's SAVE command stores the complete program (as a .COM file) with its labels and comments to disk. A commented file will naturally be larger than the same file without the comments, but it will run from DOS as any other .COM file will. If it is later reloaded with the LOAD command, all labels and comments will be recognized by AT86 and listed during the Unassemble and Trace modes.

Type the command "&varname" to see the offset of a particular variable. AT86 will then display the information: "V:013a" showing that the variable varname begins at offset 13Ah in the DS.

4. The Unassemble Mode

Files may be loaded into AT86 and unassembled with the "U beg_addr end_addr" command. Unless otherwise specified, unassembly begins at CS:100 for .COM files or CS:IP for .EXE files. A section of code may also be marked in the Display mode and then unassembled.

In unassemble mode, a new line of unassembled code is shown onscreen with each press of the space bar. Pressing PgDn will show one page of code at a time. PgUp can be used to work backwards through the program.

This unassembled file may also be written to disk with AT86's "n<filespec" option. A simple BASIC program (STRIP.BAS) is provided to strip the first 24 bytes (containing the address and opcodes for each instruction) from each line. The output from this program can then be read into AT86 to regenerate the .COM file in a manner similar to that used by the BASIC interpreter.

5. The Calculator Mode

AT86 contains on-line calculator facilities which can convert hex to decimal and binary numbers as well as perform arithmetic functions on any combinations of hex, decimal, or binary numbers. If an 8087 chip is present, AT86's functions can be used to drive a second calculator that can perform regular arithmetic and trig, hyperbolic, exponential, and other functions. When AT86 is loaded in its memory-resident mode, the results of the calculations may be captured and inserted into text.

6. The Trace Mode

A picture is worth a thousand words, the saying goes. AT86 provides not only pictures, but moving pictures of your code in operation. As the AT86 manual states, the Trace command is central to the program and provides a "dramatic" visual trace of program execution with full screen display of machine state and program instruction. I'd have to say that it's not only dramatic, but breathtaking to watch AT86 flash through your code at 1/10,000th the native speed of the CPU. It's certainly a lot better than most late night TV!

After loading a .COM file, enter the "T" (trace) command. Registers and flags are displayed at the top of the screen. The next several lines display the opcode with the current position of the instruction pointer highlighted in a reverse video bar. The contents of the stack are shown at the right of the screen. Memory affected by the pro-

HELP

123 STYLE HELP

is at hand

HELP/Control™ - an on-line help subsystem for the IBM-PC. Increases the value of your software. Save development time and money.

HELP/Runtime. A few simple subroutine calls add context sensitive on-line help to your application. **HELP/Runtime** includes tested interfaces for Microsoft C, Lattice C, Turbo Pascal, IBM BASIC (Interpreter and Compiler), Microsoft FORTRAN, IBM COBOL and assembler. It is distributed with demonstration programs in each language.

HELP/Popup. Add a powerful help system to existing applications, even in dBase or 123, without reprogramming, even without a programmer. It may be memory resident, or, installed with an application, it terminates when the application exits, releasing its memory.

HELP/Generation. Use your favorite editor and our concise screen definition language to build your help files. Compile them into a help system usable by either **HELP/Runtime** or **HELP/Popup.** The package includes sources for sample help files illustrating such features as full-sized or windowed screens.

HELP/Convenience. The screens include highlighted captions. The user selects a caption with the cursor control keys and advances to a new screen, just as with 123.

HELP/Documentation. A detailed manual, both on-line and printed, for the documentation writer and programmer includes instructions which may be incorporated into the user manual.

HELP/Environment. PC-DOS 2.0 or greater is required. **HELP/Runtime** requires approximately 9K for code and buffers for full size help screens.

HELP/Pricing. The complete package (software, both manuals, and demo programs) costs $125.00 and includes a royalty-free license to add **HELP/Runtime** to your applications and to make 25 copies of **HELP/Popup.** A demonstration diskette, including the on-line manual, costs $15.00. A free update to Release 1.1 is available to registered owners. To order, or for more information (including dealer, multiple-copy and site-license pricing) call MDS at 207/772-5436. We accept MasterCard and VISA.

MDS

MDS, INC., P.O. BOX 1237, PORTLAND, MAINE 04104

Circle Number 69 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

COMBINE THE
RAW POWER OF FORTH
WITH THE CONVENIENCE
OF CONVENTIONAL LANGUAGES

HS/FORTH

Why HS/FORTH? Not for speed alone, although it is twice as fast as other full memory Forths, with near assembly language performance when optimized. Not even because it gives MANY more functions per byte than any other Forth. Not because you can run all DOS commands plus COM and EXE programs from within HS/FORTH. Not because you can single step, trace, decompile & dissassemble. Not for the complete syntax checking 8086/8087/80186 assembler & optimizer. Nor for the fast 9 digit software floating point or lightning 18 digit 8087 math pack. Not for the half megabyte LINEAR address space for quick access arrays. Not for complete music, sound effects & graphics support. Nor the efficient string functions. Not for unrivaled disk flexibility — including traditional Forth screens (sectored or in files) or free format files, all with full screen editors. Not even because I/O is as easy, but far more powerful, than even Basic. Just redirect the character input and/ or output stream anywhere — display, keyboard, printer or com port, file, or even a memory buffer. You could even transfer control of your entire computer to a terminal thousands of miles away with a simple >COM <COM pair. Even though a few of these reasons might be sufficient, the real reason is that we don't avoid the objections to Forth — WE ELIMINATE THEM!

Public domain products may be cheap; but your time isn't. Don't shortchange yourself. Use the best. Use it now!

HS/FORTH, complete system: $395. with "FORTH: A Text & Reference" by Kelly and Spies, Prentice-Hall and "The HS/FORTH Supplement" by Kelly and Callahan

Visa Mastercard

HARVARD
SOFTWORKS
PO BOX 69
SPRINGBORO, OH 45066
(513) 748-0390

Circle Number 42 on the Reader Service Card

gram can also be displayed in a window (of variable size) at the bottom of the monitor.

With each press of the space bar, AT86 executes the current instruction and moves the highlighted IP bar to the next line. The registers, stack, and memory window are all updated. If a push instruction has been executed, the stack will be pushed down by one line, with the newest value at the top. If a conditional jump instruction is to be made, an arrow at the end of the reverse video IP bar will indicate the direction of the jump. If the contents of a memory register are to be referenced, the current contents of that memory location are shown at the top right of the screen. The execution of the program can be followed step by step simply by pressing the space bar.

To watch the program execute automatically, just hit the "C" key. AT86 begins operation of the program and runs it at 1/10,000th the normal operating speed, with the reverse video IP bar moving up and down the screen as it follows the instructions in the program. In the meantime, all the registers and the stack are continuously refreshed—it's very much like watching a complex piece of machinery in operation and is simply dazzling! Pressing the space bar stops the execution and returns to single step operation. If you want to save the screen output of the program being traced, AT86 provides a command which will redirect all the program's screen output to a user defined buffer. Typing "V" during the trace mode will toggle between the trace and program output screens.

Other Trace Mode Commands

Additonal commands available in trace mode include

B —executes the instruction at CS:IP, then sets a breakpoint to stop execution at the next encounter.
—sets # of iterations before a break.
G —accepts breakpoint address while remaining in trace mode.
Q —traces quietly and continuously (updates registers only).
F —executes quickly to next instruction, works through a loop quickly.
D —tags a subroutine for fast execution. Up to 10 may be set this way at one time.
S —single steps next instruction. Allows trace of an INT instruction, also resets a subroutine from a D tag.
N —skips or "NOPs" the current instruction.
J —JMP, overrides a conditional jump.
R —redraws screen with instruction at top.
W—toggles a window near bottom of screen for limited standard DOS screen output.
X —displays window at bottom of screen showing selected memory in either 16-byte HEX/ASCII or 64-byte ASCII format. When combined with the tracking option "T," displays and updates the area of memory surrounding the address being referenced by the program.
T —toggles tracking feature described above.
U —Undoes or backtraces. Traces instructions backwards through your code up to 20 steps or until an INT is encountered. The number of backward steps can be increased when AT86 is invoked.
V —displays the traced program's video output if the built-in "screen save" is activated. The trace screen is restored on hitting any key.
7 —toggles display of 8087 register stack.
Z —represents all the 8087 and 80287 registers in more detail.

Other Features

AT86 supports all the commands familiar to DEBUG users, but manages to enhance each one with extra features. A brief summary of some of these enhancements and other functions follows.

- Super Trace allows the user to set aside an extra 40h byte section of code that is executed following each single step of the program being traced. The extra code can also reference other code at specified addresses. This feature permits timing tests, profiling, etc., to be inserted for debugging purposes.
- Breakpoints: Up to ten permanent breakpoints may be set before tracing a program. These breakpoints may be listed in a special window, enabled, disabled, or cleared at the user's discretion. As mentioned above, a breakpoint may be set to allow a specified number of iterations to occur before it becomes effective.
- Data in either byte or word form may be output to any specified port address. Incoming data from any specified port can be read and will be displayed in both hex and binary format.
- Baud rates and COM address may be set from AT86.
- A data entry mode is also available (similar to DEBUG's E command), which permits insertion of single byte values

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

#1 C interpreter

C-terp

The professional C development environment

Your C compiler creates great final code . . . but as a programming tool, it's too, too slow.

With C-terp you can edit, debug, and run without the wait. Nothing, but nothing, is faster for developing professional C programs.

Choose the perfect C-terp companion for your C compiler

C-terp/Microsoft	C-terp/XENIX
C-terp/Lattice	C-terp/Aztec
C-terp/Mark Williams	C-terp/C86

Link in all your compiler's functions, your own functions, add-on libraries, assembly routines, and data objects. Get instant access to everything in the C-terp interactive environment.

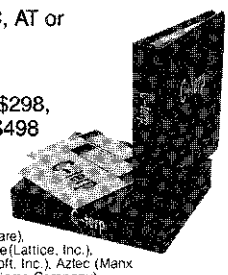
Only C-terp offers all this and more

- **Full K&R with common ANSI enhancements**
- **Source level interactive debugging**
- **Software paging for your big jobs**
- **Complete multi-module support**
- **Run-time pointer checking**
- **Unsurpassed reconfigurable screen editor**
- **Dual display and full graphics support**
- **Large model** ■ **Call-in**

ORDER C-terp TODAY (specify compiler)

C-terp runs on IBM PC, AT or compatibles.

Price:
MS-DOS 2.x and up - $298,
Xenix System V 286 - $498
MC, VISA, COD
30-day money-back GUARANTEE



Trademarks: C-terp (Gimpel Software), C86 (Computer Innovations), Lattice(Lattice, Inc.), Xenix, Microsoft, MS-DOS (Microsoft, Inc.), Aztec (Manx Software), Mark Williams (Mark Williams Company), IBM (International Business Machines, Inc.)

GIMPEL SOFTWARE

3207 Hogarth Lane, Collegeville, PA 19426

(215) 584-4261

Circle Number 11 on the Reader Service Card

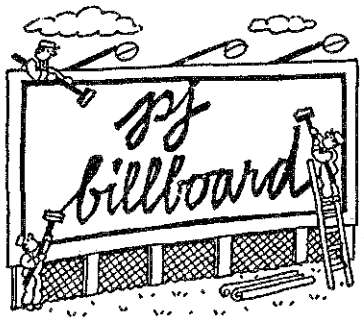
directly into memory. The backspace key lets you re-enter an erroneous value. Special structure templates can be created to permit reading of data structures in their natural format (including linked lists and data structures with mixed data types).

- DEBUG's fill command is also enhanced, letting you fill a specified range with a marked block of code or fill a range with code using the "A" (assemble) function.
- The GO command supports conditional breakpoints and the special features of the advanced trace command.
- Keyboard scan codes for any key or key combination may be read and displayed onscreen using the "KI" command.
- The screen colors, video buffer addresses, interrupt controller mask, and window sizes may be configured and set either for one session or as default values.
- For machines with 80286es, AT86 permits entry to real mode and access of as much as 16 megabytes of memory in protected mode.
- A specific vector or DOS interrupt may be accessed directly from the command mode without resetting current registers.

These and other features make Advanced Trace86 a valuable debugging tool for the advanced programmer and possibly the best assembly language teaching device available for the 8086/80286 microcomputer family. This is one package that anyone interested in assembly language should find very worthwhile. □

Readers may contact Peter Feldman

c/o IC Consultants
816 State Street, Suite "A"
Santa Barbara, CA 93101
(805) 564-3020



VITAMIN C

It's good for your system.

FULLY Integrated, Data Entry Windows!

- Complete input formatting
- Unlimited Validation
- Full attribute control
- Multiple virtual windows
- Fully automatic, collision proof overlay and restore
- Print to & scroll background windows
- Animated window "zoom"
- Move, grow, shrink, hide, or show any window
- "Loop function" allows processing while awaiting input

AND MUCH MORE!

$149.95

Includes 100% source, tutorial, reference manual, examples, and sample programs.
Specify Microsoft, Lattice v2 or v3, Computer Innovations, Aztec, DeSmet, or Mark Williams. Ask about Unix.

100% Money Back Guarantee

NOW ... VCScreen!
Our new interactive screen "painter" actually lets you draw your data entry windows! Define fields, text, boxes & borders. Move them around. Change attributes. Then the touch of a button generates C source code calls to the Vitamin C routines!

$99.95

Requires Vitamin C Library above. For IBM & compatibles.

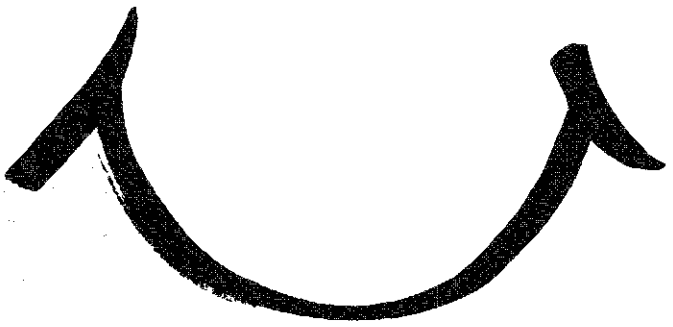
For Orders Or Information, (214) 245-6090

Creative Programming Consultants, Inc.
Box 112097 Carrollton, TX 75011-2097
Include $3 ground, $6 air, $15 overnight shipping. $25 if outside USA. Texans add 6⅛% tax. All funds must be in U.S. dollars drawn on a U.S. bank.

creative PROGRAMMING

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

COMPUTER FAIRES® GIVE YOU AN UNFAIR ADVANTAGE.



The Northeast and West Coast Computer Faires are going to help you surge ahead into new business opportunities in 1987. Attendees will find the latest and most innovative products and services to put them ahead in the quest for productivity. Exhibitors searching for a sophisticated audience that's ready to buy will be smiling at their extra sales.

And extensive promotion, a full conference series, and special features like State-of-the-Art Professional Development Seminars help add up to give you the Computer Faire advantage—a smile of triumph.

Attention exhibitors! Call (617) 449-6600, Ext. 4014, now to reserve exhibit space or to receive more information.

West Coast Computer Faire '87
March 26–29, 1987, Moscone Center, San Francisco

Northeast Computer Faire '87
October 15–17, 1987, World Trade Center, Boston

I want the Computer Faire advantage.

Send me ☐ exhibitor ☐ attendee information on ☐ Northeast ☐ West Coast Computer Faires

Name _____ PG1186

Title _____

Company _____

City _____ State _____ Zip _____

Telephone ( ) _____

Return to The Interface Group, Inc., 300 First Avenue, Needham, MA 02194

Circle Number **58** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

PJ Billboard is a product and services directory containing listings for hardware and software of interest to the IBM PC programmer. The goal is to provide our readers with a wide range of information on current products and updates. PJ Billboard serves our advertisers by providing a low cost, flexible format in which they can present their products to our readers.

The standard layout is four columns, 9 picas wide, with copy set in 7 point on 8 point leading. PJ Billboard offers a 400-character product description, including your company name, address, and telephone number. Boldface and italics are available.

The listings are $65, prepaid, non-commissionable for a one-time insertion. Listings must be received typed, preferably on one of our convenient order forms. Copy deadlines are the same as the normal ad materials closing dates, which can be found on our rate card, or you may call the Programmer's Journal Advertising Department at 503-484-0602 for more information.

Send for Free Information
An advanced book for Turbo Pascal Programmers. NOT A BEGINNER'S TUTORIAL! In-depth, thorough, practical, technical. Chapters written by Turbo Pascal experts. Topics include: Optimization techniques, using interrupts, build a Pascal subset compiler, understand bit mapped graphics, and more.

Rockland Publishing
190 Sullivan, Ste. 109
Columbia Falls, MT 59912

UNDER-C™ LIBRARY
With IBM-PC & UNDER-C™ LIBRARY (supports Microsoft C [Version 2], Lattice C; Instant-C; Comp. Innov. C86; DeSmet) access DOS & BIOS functions. Utility programs written in C (includes MAKE and print); screen generator; source codes; not copy protected; no royalties. Manual and 4 floppies $95. Ohio residents add 6-1/2% sales tax.

Quayle Research, Inc.
6548 Edgerton Rd.
N. Royalton, OH 44133

THE SCREEN MANAGER
A complete screen management system for the PC. Up to 15 simultaneous windows. Complete control over text and attributes of every character in every window. Much more.

Comprehensive manual. Sample programs. IBM PC or true compatible.

MASM + MS C V3 + FORTRAN V3.3 or MASM + Lattice C. (Others coming). $149.95 ppd. VISA/MC. Outside USA add $10.00. NV res. add sales tax.

CONCEPT SOFTWARE
P.O. Box 26981
Las Vegas, NV 89126

PageMaker Services
User manuals, Advertising, Packaging and Direct Mail
- Graphic production
- Design and illustration
- Editing and copy writing
- Download your existing documents
- Professional quality work
- Fast turnaround

Ralph Bentley
Blackbird Studios
888 Armstrong Ave.
Eugene, OR 97404
(503) 689-6651

FINALLY!
A Subroutine Library for Compiled BASIC
Load directories into arrays, WINDOWING, Graphics pie and bar charts, 150 more routines to add power to Microsoft QuickBASIC or IBM BASIC Compiler 2.0
30-Day Money-Back Guarantee.
VISA/MC/Check/MO
FINALLY! is only $99.00 + $4.00 S&H

KOMPUTERWERK, INC., DEPT PJB
851 Parkview Blvd.
Pittsburgh, PA 15215
(412) 872-0384

Documentation
- Writing, editing, formatting for photocopying or typesetting
- MS DOS, Apple, AT&T 3B series UNIX
- Over thirty manuals
- Credits include
  Dave Winfield's BATTER UP!
  (Billboard Top Twenty)
  Action Inn Management
  (AT&T 3B2)
  SmartCAM
  (MS DOS, Apple)
  Programmer's Journal
  (Managing Editor)

Erin O'Connor
c/o Programmer's Journal
PO. Box 30160
Eugene, OR 97403
(503) 484-2162

IMPORTANT TAX INFORMATION!

WARNING—Under the new tax reform laws, miscellaneous expenses (like the cost of your **PJ** subscription) will have to be combined in 1987, and you'll only be allowed to deduct the portion that exceeds 2% of your adjusted gross income.

WHAT YOU SHOULD DO—To save the most money, choose our two- or three-year option when you subscribe or renew. You'll save 20% to 25% off **PJ**'s newsstand price and be able to deduct the full amount *in this tax year*.

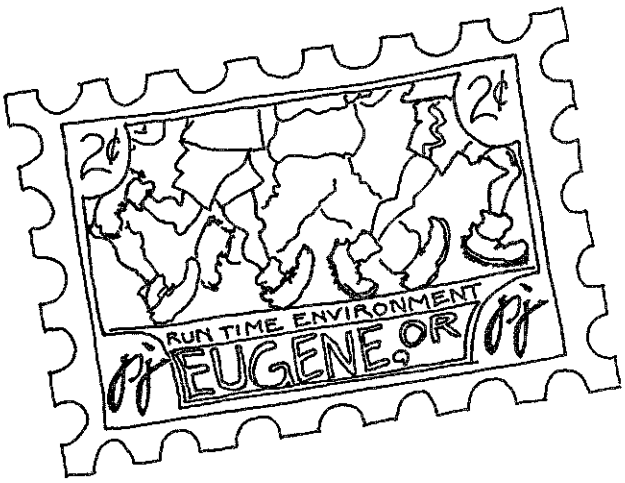
PLAY IT SMART—**PJ**'s prices have never been lower and the timing has never been better. Choose an extended term and send in your payment today. You'll save money now and again when you file your Federal Income Tax Return.

Sincerely,

Robert M. Tappan
Circulation Manager

P.S. Remember—to qualify for the full deduction, you must make your payment in 1986, so send in your check or VISA/MC information today.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



PEER REVIEW

Rank the articles in this issue of PJ. Refer to the list below for the article letters, and grade the articles on the postage paid Reader Service Card, Number 1 the best, and so on.

A _____ PJ FORUM: DTACK GROUNDED, The Junk Mail Flyer, Hal W. Hardenbergh

B _____ REDIRECTIONS: Unix Books, Part 1, Frank D. Greco

C _____ REDIRECTIONS: How Many Amoebas, M.J. Budlong

D _____ DOS: Parents, Children, Redirection, and Piping, George Defenbaugh

E _____ DOS: Terminate and Stay Resident, M. Steven Baker

F _____ PASCAL: Turbo Pascal Inline Code, James M. Smith

G _____ PASCAL: Assembler Source Code for DOS Function Calls, Stephen K. O'Brien and Wing Kit Cheung

H _____ C: Alignment Problems and Managing Memory Allocation, William Hunt

I _____ C: The Proposed ANSI C Standard, Rex Jaeschke

J _____ LASER PRINTERS: The Canon A1 Laser Printer, Paul Barkley

K _____ REVIEW: Genifer: A Powerful dBASE Applications Generator, Paul Barkley

L _____ REVIEW: Advanced Trace86, Peter Feldman

ADVERTISER INDEX

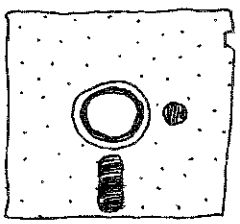
CIRCLE NUMBER		PAGE NUMBER
52	Aldebaran	43
53	Allen, Emerson & Franklin	17
1	ATRON	42
30	Austin Code Works	49
84	Beagle Bros.	12
2	Blaise Computing	37
3	Borland International	9
38	C-Ware	42
64	Chalcedony Software	14
56	Compuview Products, Inc.	5
*	Creative Programming Consultants	61
83	Crescent Software	55
66	Custom Software Systems	17
5	Dair Computer Systems	35
41	Design Solutions	35
67	E-Heart Engineering	49
8	Ecosoft, Inc.	Cov 4
9	Entelekon	56
11	Gimple Software	61
42	Harvard Softworks	60
57	Hawkeye Grafix	24
58	Interface Group	62
59	L.U.C.I.D.	11
63	Laboratory Microsystems, Inc.	10
44	Lahey Computer Systems	29
81	Lifeboat	6
69	MDS, Inc.	59
13	Micromotion	38
65	Mnematics Videotex	51
15	Oliver Computing	43
16	Opt-Tech Data Processing	30
71	Pecan Software Systems, Inc.	12
6	Periscope Company, The	34
26	Polytron	57
31	Quantumn Information Systems	11
46	Quilt Computing	51
20	Rational Systems	53
33	Soft Advances	19
47	Solution Systems	31
72	Speedware	53
82	Stamey Systems	28
85	Sunny Hill Software	45,47
35	Turbo Source Search	19
73	Whitewater Group	Cov 2

*This advertiser prefers to be contacted directly; see ad for address and telephone number.

To receive advertising information, circle #89

PUT IN A GOOD WORD

When you contact these advertisers for information, please be sure to mention that you saw their product advertised in **Programmer's Journal**. Thanks!

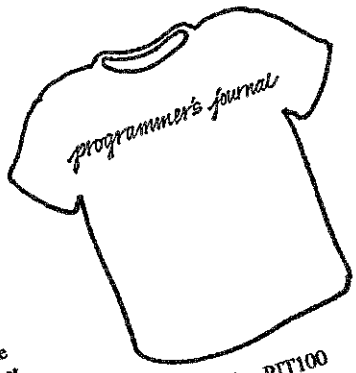


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

T-SHIRTS

BORN TO PROGRAM PJT101

Show your true colors when cruisin' the local computer store. Let 'em know that programmers rule in an authentic black T-shirt with this hot multi-color design.



PJ LOGO PJT100

For those of you who care enough to wear the very best, we offer our world famous PJ logo printed on the T-Shirt color of your choice: White, Lt. Blue, Beige or Gray.

PARALLEL PORTS PJT102

These twin vessels are guaranteed to enhance communications with your favorite printer or anyone else you dump your text to. Multi-color imprint on a wine-colored T-Shirt.

NOW! SOFT WEAR FROM PROGRAMMER'S JOURNAL!
Only $9.95 each, postage paid.

By popular demand, your favorite computer jokes and cover graphics by Julia King and Ralph Bentley come alive in dynamic multi-color imprints on high quality 50/50 cotton T-Shirts. Use the form on this page to order your own unique, custom soft wear. Fill out and mail today with your payment to :

Programmer's Journal
29 W. 29th Ave., Suite 5
Eugene, OR 97405

RS-232 HEAD PJT103

Here's a connection you can't refuse. Plug your head into this neat-as-a-pin computer gray T-shirt with colorful high-tech imprint.

COMMON C BUG PJT104

Everyone's got 'em! Now proudly display your own specimen. The bug is brown (naturally), crawling across a gold nameplate on a beige T-shirt.

SIZES AVAILABLE: S, M, L, XL

DESIGN NO.	QTY	COLOR	SIZE	AMOUNT	SEND TO:
					NAME
					COMPANY
					ADDRESS
					CITY STATE ZIP
					CHECK ENCLOSED ☐ CHARGE MY: VISA☐ MC☐
TOTAL ENCLOSED				$	CARD NO. EXP. DATE

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

ECOSOFT HOLIDAY SPECIALS

EVERYTHING FOR THE C PROGRAMMER

If you know someone (perhaps yourself!) that wants to learn the C programming language, have we got a deal for you. For a limited time, you get:

Eco-C88 C compiler	$59.95
CED Program editor	29.95
Developer's Library Source Code	25.00
ISAM File Handler	15.00
Flexi-Graph Graphics Package	39.95
Eco-Lib MSDOS Compatible Librarian	29.95
Ecosoft Windowing Library	29.95
C Programming Guide book (Que Corp)	20.00
C Self Study Guide book (Que Corp)	17.00
C Programmer's Library book (Que Corp)	22.00
Total:	$288.75
Holiday Special:	***$175.00***
Save Over:	$110.00!!!

The Holiday Special has everything you need for your C programming needs. Eco-C88 is a full K&R C compiler (less bit fields), with over 200 library functions, source code to our cc and mini-make utilities, and generates code rivaling compilers costing 10 times as much. You also get the source code to our library, an MSDOS-obj compatible librarian, one of the neatest graphics packages around, an easy-to-use windowing package plus a library of books to teach you the C language. An unbelievable package at an unbeatable price of only $175.00.

Microstat Statistics Package

Since 1977, Microstat has been one of the most popular statistics packages available for microcomputers. Just some of its features include:

Data Managment Subsystem	Multiple Regression (with Stepwise)
Data transformations	Correlation Analysis
Convert External data (e.g. Lotus, Dbase II, etc.)	Time Series Analysis
Descriptive Statistics	Nonparametrics
Hypothesis Testing	Probability Distributions
Crosstabs and Chi-Square	Scatterplots
ANOVA	Factorials, Permutations

Holiday Special: $250.00

plus many other unique and easy-to-use features. Microstat is available for all MSDOS machines or any CP/M80 system using a Z80 CPU.

The regular price is a bargain at $375.00. At the Holiday Special price of $250.00, it's a steal!

To take advantage of these special prices, we must have your order in our offices by Jan. 15, 1987.
To order, call or write:

1-800-952-0472 (for orders)
or
1-317-255-6476 (tech. info.)

ECOSOFT INC.

Ecosoft Inc.
6413 N. College Ave.
Indianapolis, IN 46220

Trademarks: Ecosoft: Microstat, Eco-C88; Microsoft: MSDOS; New Orleans General Data Services: Flexi-Graph

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Circle Number 8 on the Reader Service Card